Universidad Zaragoza
1542

Trabajo Fin de Grado

Estudio de QoX de usuario: Herramienta para el análisis de uso en dispositivos móviles

Study of user's QoX: Tool for the anylisis of use on mobile devices

Autora

Patricia Chueca Minguillón

Directora

Rosa Mora Marín

Ponente

Julián Fernández Navajas

Escuela de Ingeniería y Arquitectura

Curso 2020/2021

Estudio de QoX de usuario: Herramienta para el análisis de uso en dispositivos móviles

Resumen

Durante los últimos años, la tecnología móvil ha ido evolucionando desde los primeros terminales, que permitían únicamente llamadas, hasta los smartphones de hoy en día a los que podemos considerar un pequeño ordenador de bolsillo. A la evolución tecnológica le ha acompañado también un cambio en las pautas de uso de los mismos: ya no sirven únicamente para llamar, sino para ver vídeos, hacer videollamadas, navegar por internet o acceder a nuestras redes sociales.

Para este Trabajo de Fin de Grado se ha desarrollado una herramienta que permite la recogida de datos en terminales móviles para analizar el uso de los mismos, estudiar la calidad de experiencia (QoE, *Quality of Experience*) de los usuarios, e incluso ir más allá con el estudio de la QoX, siendo la X cualquier variable susceptible de ser estudiada bajo una perspectiva de calidad.

Tanto el análisis previo, como un estudio de las tecnologías que utilizaremos, han sido aplicados en el desarrollo de un sistema que permite la sincronización, sobre un servidor central, de la información recopilada en el terminal de un usuario. Por esto, se ha desarrollado una aplicación Android para obtener de manera automatizada datos de diferentes sensores de un dispositivo y enviar la información a un servidor. Además, se ha elaborado un servidor de pruebas que permita testear el correcto funcionamiento de la comunicación entre elementos dentro del sistema.

El proyecto elaborado para este trabajo es un prototipo que podría servir como base para la elaboración de una aplicación optimizada por expertos y que se pueda desplegar en un escenario real.

Study of user's QoX: Tool for the anylisis of use on mobile devices

Abstract

During the last years, mobile technology has been evolving, from the very first terminals that only allowed phone calls to nowadays smartphones, that can be considered small pocket computers. Technological evolution has also been accompanied by a change in usage guidelines: they are not only useful to call, but also to watch videos, make video-calls, browse the web or to access our social newtworks.

For this Bachelor's Degree Final Project, it's been developed a tool which allows to collect data in mobile devices to analyze it's use, to study users' quality of experience (QoE), and even go further with the study of QoX, being the X any variable that can be studied from a quality perspective.

Both the previous analysis and a study of the technologies that we will use, have been applied in the development of a system that allows synchronization, on a central server, of the information collected in the terminal of a user. Because of this, an Android application is been developed to obtain, in an automated way, data from different sensors of a device and send information to a server. Besides, a test server is been created as well to test the right communication and functionalities between system elements.

This work presents the developed prototype, that could serve as a base to create an optimized application by experts and that it can be deployed on a real scenario.

Índice general

Capítulo 1

Introducción

1.1. Introducción

Desde sus inicios en el ámbito militar y gubernamental, hasta su posterior popularización general, Internet se ha convertido en un elemento clave para las comunicaciones y el intercambio de información.

Con el paso del tiempo se ha ido produciendo un esfuerzo global para conseguir la convergencia de tecnologías de voz y datos tanto en redes, como en los diferentes elementos que las componen (módems, dispositivos, etc.). Los terminales analógicos, que únicamente tenían la función de realizar llamadas, se fueron adaptando para recibir datos y mostrarlos a los usuarios; desde mensajes cortos hasta contenido multimedia. Por otra parte, se fueron desarrollado aplicaciones y protocolos para la transmisión de voz a través de la red de datos, mientras que los terminales de datos, gracias al uso de módem, lograban transmitir información a través de los canales de voz.

A la par que la evolución tecnológica, se sucede también un cambio en los aspectos económicos y sociales. El desarrollo y la innovación pasan de ser una motivación puramente tecnológica (mejorar lo que ya existe o crear cosas nuevas), a ser el usuario y sus necesidades el centro del desarrollo. Esto da lugar a pasar del concepto QoS (*Calidad de Servicio*) centrado en la tecnología, al concepto de QoE (*Calidad de Experiencia*) centrado en el usuario; una medida subjetiva de la satisfacción que tiene el cliente sobre una aplicación o servicio. Actualmente aparece el concepto de QoX (*Calidad de X*), siendo la X una incógnita, es decir, cualquier aspecto que decidamos tratar sobre el sistema: servicio (*QoS*), experiencia (*QoE*), o cualquier otro que consideremos necesario tener en cuenta. El surgimiento de las redes sociales y nuevas formas de comunicación (mensajes o vídeollamadas), ha ocasionado un aumento de su necesidad de uso, siguiendo diferentes pautas de comportamiento y pudiendo dar lugar a adicciones en sectores de población emocionalmente más vulnerables, como pueden ser los adolescentes. Ademas de la QoE de usuario, estos diferentes grados sobre la necesidad de uso se convierten en otro aspecto interesante de estudio en la QoX. Se trata de un elemento interesante dado que desde diferentes puntos de vista (por ejemplo, marketing y educación), puede suponer aspectos enfrentados o incluso intereses contrapuestos, un conflicto de prioridades que se debería resolver.

En definitiva, con el fin de mejorar la QoX aplicada a las necesidades de uso, es necesario conocer las pautas de uso, comportamiento y grado de satisfacción de los usuarios con respecto a las mismas, dando a profesionales con diferentes puntos de vista (educativo, marketing, etc), la posibilidad de analizar diferentes situaciones en cuanto al uso y poder perfilar distintos grupos de usuarios (por edad, sexo, aficiones, formación, ...). Además, re-

sulta interesante disponer de la capacidad de actuación por parte de grupos expertos en un futuro, y en tiempo real, ante situaciones que representen un riesgo para el usuario (posibles trastornos de comportamiento en el uso, riesgos de seguridad, deterioro de la calidad percibida, etc.).

1.2. Motivación y objetivos

Partimos de la oportunidad de estudiar y desarrollar un prototipo sobre un caso real, la problemática comentada en la introducción. La idea de este Trabajo Fin de Grado surge de la necesidad de estudio, por parte de investigadores de la Facultad de Educación de la Universidad de Zaragoza, de la calidad de experiencia (QoE) y la QoX respecto a las pautas de comportamiento de los diferentes grupos de usuarios.

Tal y como se ha comentado en la introducción, el aumento en la necesidad de uso de las nuevas tecnologías ha supuesto la aparición de vulnerabilidades relacionadas, sobre todo en los jóvenes. En la actualidad, la principal vía de estudio sobre el comportamiento de los usuarios es mediante cuestionarios, lo cual supone un conocimiento no completo de la situación real, dado el carácter subjetivo de los resultados obtenidos.

Con este proyecto, nos centramos en la captura de información (requerida y pre-definida por los expertos) de uso en los terminales de usuario. Además de la captura de información, debe permitir la actuación de los superusuarios (expertos) sobre el terminal en función del comportamiento detectado en el mismo.

De esta manera, este trabajo pretende desarrollar una herramienta que permita la obtención de datos objetivos para completar el análisis de uso real de los dispositivos con el fin de conocer posibles situaciones que representen un riesgo para el usuario. Además, garantizando que la recogida de datos sea anónima y segura, y, en un futuro, estar en posición de facilitar la actuación al respecto en tiempo real.

En la *Figura 1.1* se muestra un esquema del sistema que se pretende realizar con este trabajo, en el cual los dispositivos conectados a diferentes redes puedan recopilar datos y enviarlos a distintos servidores, que se encargarán de recibirlos y procesarlos convenientemente e incluso podrán actuar sobre los terminales deseados para limitar su funcionalidad y modificar el comportamiento de los usuarios. En el capítulo de *Sistema desarrollado* se especificará el comportamiento y características de cada uno de estos elementos.

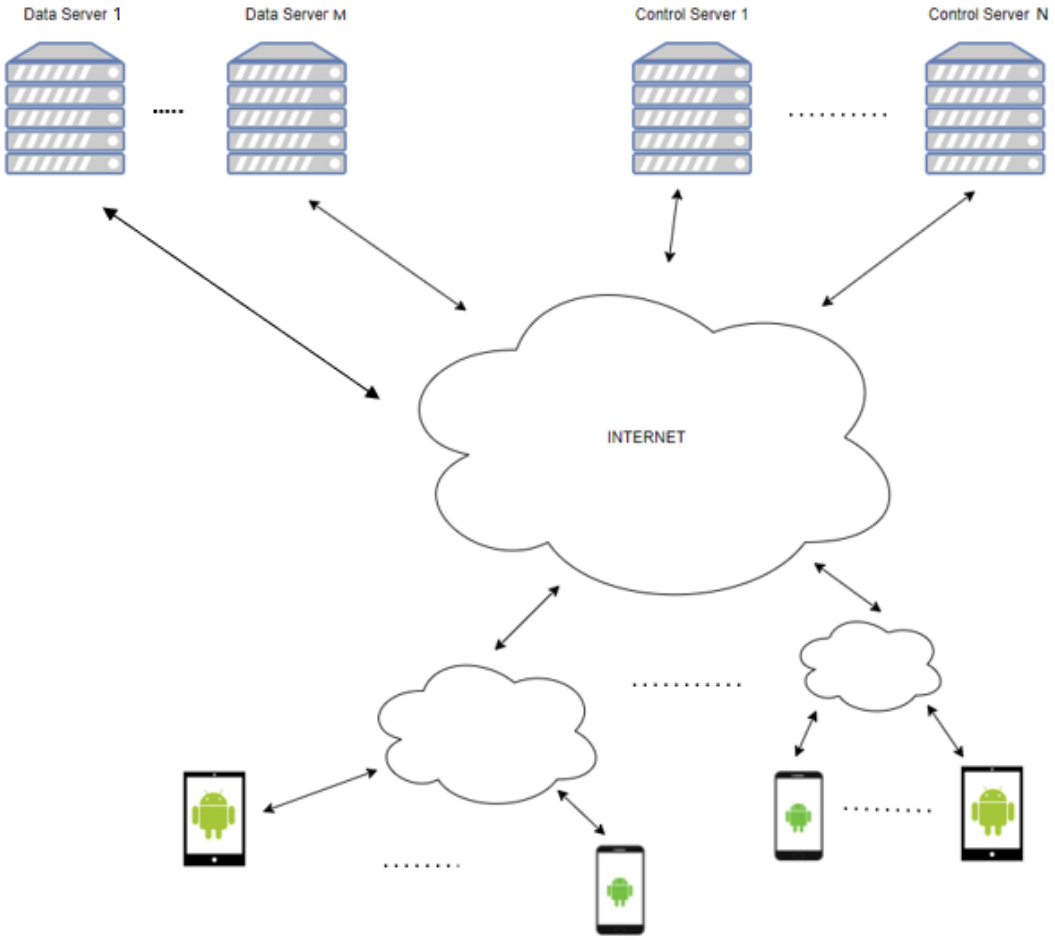


Figura 1.1: Arquitectura planteada del sistema

1.3. Organización de la memoria

La memoria se ha organizador con la siguiente disposición:

1. Capítulo 1: se presenta la descripción del trabajo, posibles problemáticas y los objetivos que se pretenden alcanzar.
2. Capítulo 2: se expone el planteamiento y análisis previo así como las herramientas que se han utilizado y por qué.
3. Capítulo 3: se explican las diferentes partes del desarrollo y funcionamiento del sistema.
4. Capítulo 4: se repasan las pruebas realizadas para la comprobación del correcto funcionamiento de la aplicación.
5. Capítulo 5: se plantean las conclusiones obtenidas y las líneas futuras.

Finalmente se incluyen anexos al final del documento:

- Instalación y configuración de herramientas para el desarrollo
- Implementación de aplicación móvil Android

Capítulo 2

Análisis Previo

2.1. Planteamiento

Uno de los puntos de partida que se han realizado para el desarrollo de este trabajo ha sido la definición de las herramientas que se utilizarán para llevarlo a cabo.

Dada la amplitud de opciones y marcas que existen en la tecnología móvil (Huawei, Samsung, Xiaomi, Apple...), se ha optado por elegir entre los dos sistemas operativos más extendidos entre los dispositivos del mercado: Android e iOS.

La elección del SO, sobre el que se va a llevar a cabo el desarrollo, condiciona también el lenguaje de programación y las herramientas a utilizar. Además, se ha de tener en cuenta si se va a realizar una aplicación nativa, es decir, exclusiva para un SO concreto, o una aplicación híbrida, que permite la implantación en diferentes SO. Estas últimas aplicaciones se descartan como posibilidad para este trabajo, dado que están más destinadas a la adaptación móvil de aplicaciones web, por lo que no permitirían la recogida de algunos datos requeridos.

Una vez decidido que se va a desarrollar una aplicación nativa y los dos sistemas operativos que se van a utilizar, se plantean diferentes posibilidades para cada uno de ellos.

1. **Android**
    - Lenguajes: *Java* y *Kotlin*
    - Plataformas de desarrollo: *Android Studio*
2. **iOS**
    - Lenguajes: *Objective-C* y *Swift*
    - Plataformas de desarrollo: *Xcode*

Respecto a la comunicación e intercambio de información de los terminales con el servidor se contemplan distintas arquitecturas y protocolos (*HTTP, TCP, COAP, etc.*):

- HTTP (*Hyper Text Transfer Protocol*): Protocolo de nivel de aplicación que permite comunicar datos a través de internet. Usado principalmente para la interacción con aplicaciones web.
- TCP (*Transmission Control Protocol*): Protocolo de nivel de transporte que permite conectar dispositivos de red a internet. Este protocolo proporciona comunicación entre una aplicación y el protocolo IP.

- UDP: (*User Datagram Protocol*): Protocolo similar a TCP. No requiere el pre-establecimiento de conexión ni incluye confirmación de entrega del datagrama.
- COAP (*Constrained Application Protocol*): Protocolo de nivel de aplicación para la comunicación de dispositivos electrónicos simples sobre internet. Pensado principalmente para sensores de baja potencia y diseñado para incluir al modelo HTTP otros requisitos (multicast, simplicidad y bajo overhead).

Finalmente, se ha optado por una arquitectura básica cliente-servidor sobre TCP.

Para la creación de un servidor básico para pruebas, únicamente es necesario un editor de código (*Visual Studio Code, Notepad++, SublimeText...*) que permita simular la interacción cliente-servidor.

Como se explicará en capítulos posteriores, uno de los problemas que pueden surgir a la hora de capturar información de un dispositivo es que esté limitada por la seguridad del propio SO, o que la propia comunicación esté ofuscada, por lo que no se pueda obtener información sobre las conexiones que se llevan a cabo. Otra forma posible de conocer esta información es realizando un análisis de las conexiones IP que lleva a cabo el dispositivo con la red, conociendo, por ejemplo, los servicios a los que accede (o al menos, sus proveedores) a partir del análisis temporal de las conexiones a direcciones IP previamente conocidas. De los proveedores podríamos obtener la información deseada. Es por ello que se plantea también el uso de un analizador de captura de tráfico en el sistema. Esta captura se podrá llevar a cabo o bien en el propio terminal, si éste lo permite, o bien en un equipo externo que intercepte las comunicaciones del terminal. En el caso de equipos externos podrá implementarse, si el acceso del terminal es WiFi, capturando en el propio medio inalámbrico o en la red fija que da acceso a Internet cuando se tenga acceso ésta.

En el siguiente apartado se enumeran las herramientas finalmente escogidas, sus características y los motivos de su selección.

2.2. Herramientas

A continuación se enumeran las herramientas, lenguajes de programación y plataformas utilizadas para este proyecto, cuya instalación y configuración se detallarán en el *Anexo 1*:

- **Android Studio:** Entorno de desarrollo integrado (IDE) oficial para el sistema operativo Android proporcionado por Google. Soporta lenguaje *Java* y *Kotlin.* Android Studio proporciona una herramienta muy útil para poder realizar pruebas: *AVD Manager.* Esta herramienta se trata de un gestor de dispositivos virtuales Android (emuladores), es decir, un simulador que nos permite elegir entre dipositivos (TV, móvil, smartwatches, tablets y una adaptación para vehículos) compatibles con este sistema operativo. Empleamos este entorno para crear la aplicación y hacer pruebas en un dispositivo virtual. Se ha escogido Android y, por tanto, este entorno, debido a la disposición de un terminal Android para hacer pruebas.

- **Visual Studio Code:** Editor de código fuente multiplataforma desarrollado por Microsoft. Incluye soporte para la depuración, control integrado de Git, resaltado de sintaxis, finalización inteligente de código, fragmentos y refactorización de código. Este programa está ampliamente extendido para el desarrollo, además de proporcionar herramientas como terminal integrado, biblioteca de extensiones y soporte para múltiples lenguajes y SO.

- **Wireshark:** Software para la captura y análisis de tráfico de red. Programa con interfaz gráfica que permite analizar los paquetes que se intercambian diferentes protocolos por la red. Herramienta sencilla y práctica, dispone de entorno gráfico y se trata de una herramienta ya conocida. Permite un análisis extra del uso del dispositivo gracias a la captura de información que el terminal intercambia con la red.

- **Java:** Lenguaje de programación de propósito general, orientado a objetos y que permite su ejecución independientemente de la arquitectura. Hace uso de una máquina virtual java (JVM) para ejecutarse. Se utiliza para el desarrollo de aplicaciones cliente-servidor. Lenguaje escogido debido a la familiarización con el mismo. Usado para el desarrollo de la lógica de la programación.

- **XML (*eXtensible Markup Languaje*):** Lenguaje de etiquetas extensibles. Se emplea para describir datos y permite el diseño de la IU (*Interfaz de usuario*). Este lenguaje va unido al uso de java en el desarrollo de la aplicación, en el caso del uso de *Kotlin* no es necesario.

- **TCP/IP:** Modelo de comunicación en redes basado en el modelo teórico OSI y que hace referencia a un grupo de protocolos:

  - TCP (*Transfer Control Protocol*): Protocolo de transmisión que permite la conexión e intercambio de datos entre dos elementos de red.
  - IP (*Internet Protocol*): Protocolo de internet. Su función principal es el uso bidireccional, en origen o destino, de comunicación para transmitir datos mediante un protocolo no orientado a conexión que transfiere paquetes conmutados a través de distintas redes físicas, previamente enlazadas según la norma OSI de enlace de datos.

  Este modelo proporciona una comunicación segura entre cliente y servidor y posee un alto grado de fiabilidad. Además, es compatible con las herramientas estándar de análisis de red.

La *Figura 2.1* muestra en qué parte del sistema se localizan cada una de las herramientas. En la parte derecha de la imagen, encontramos la arquitectura general del sistema, en la que se emplea el analizador de red (*Wireshark*) para capturar tráfico. Por otro lado se muestran las conexiones entre los terminales y la red, y la red y los servidores, empleando el modelo TCP/IP. En la parte inferior se representan los dispositivos *Android* (tablets, móviles, Chromebooks, etc.) para los que se ha diseñado este trabajo. Finalmente y en la parte izquierda, recuadrado en color naranja, aparece el *Test Environment* o entorno de pruebas, que tendrá un esquema similar a la arquitectura general. Por otro lado, en este caso, el *Test Server* estará implementado dentro del programa *VSCode*. Este servidor consistirá

en un script con una clase *Java* que simulará el comportamiento del servidor de datos (comunicación a través de sockets TCP, lectura de datos y respuesta a mensajes establecidos por el protocolo de comunicación).

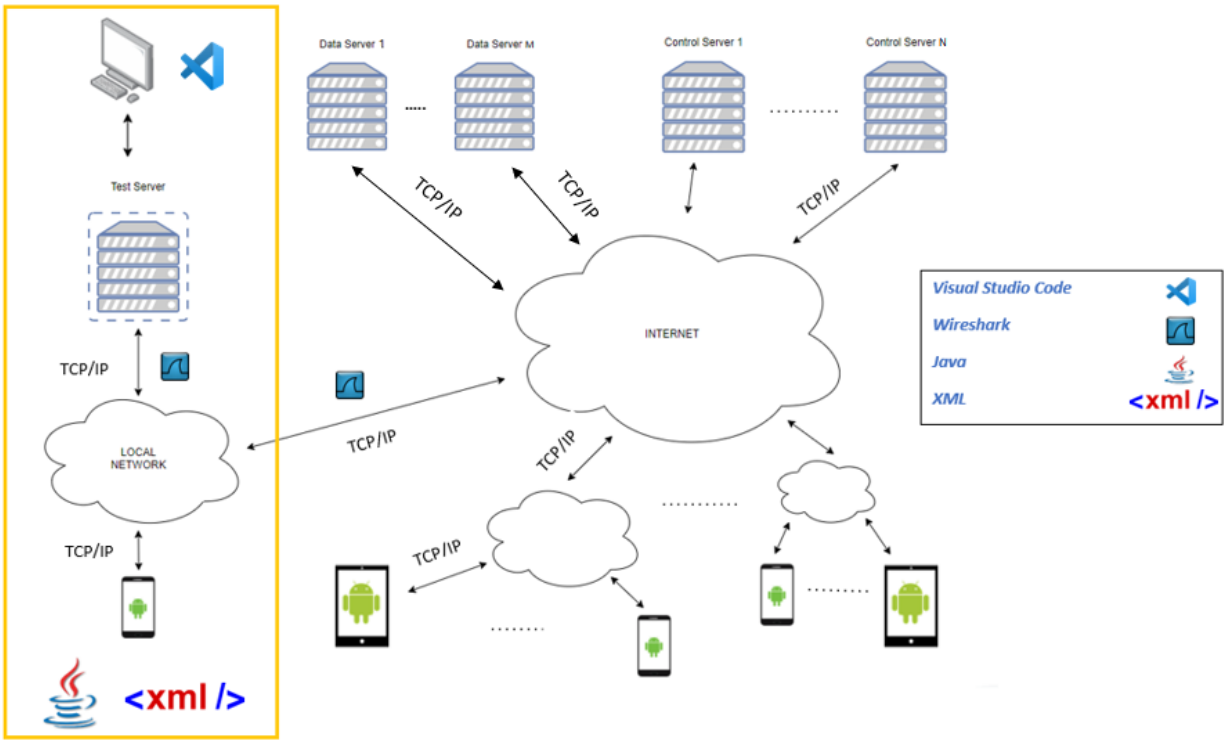


Figura 2.1: Localización de herramientas en el sistema

Capítulo 3

Sistema desarrollado

3.1. Planteamiento General

El sistema que se ha planteado es una herramienta prototipo, que permite dar apoyo a los requerimientos del usuario experto y las necesidades que nos ha planteado. Aunque inicialmente se trate de un sistema básico, existe la posibilidad de ampliación futura a partir de la optimización y crecimiento del código, así como en el uso de potenciales herramientas y la propia colaboración de los usuarios finales.

Para este prototipo se ha elaborado un protocolo básico de comunicación entre terminales Android y servidores de datos y control con cinco mensajes, y una serie de características de usuario que podrían también modificarse, o ampliarse, para dar solución a futuras necesidades. Aunque en este apartado contemplamos la existencia de mensajes que permitan la actuación sobre los terminales, en líneas futuras será necesario detallar diferentes tipos y definir su estructura.

3.2. Protocolo de comunicación

En el desarrollo de esta herramienta se ha creado un protocolo de comunicación basado en la arquitectura cliente-servidor de TCP/IP. Para ello, se han definido cinco mensajes tipo y sus correspondientes respuestas:

ID	Tipo	Origen
1	Registro	Terminal
2	Datos	Terminal
3	Login	Terminal
4	Actualización de datos	Terminal
5	Información de control	Servidor de control

Cuadro 3.1: Tabla resumen tipos de datos

En la definición de la información que portará cada uno de los mensajes ha sido necesario tener en cuenta que se garantice la anonimidad y confidencialidad del usuario. Aunque en algunos de los mensajes sí que se recoge información personal (edad, sexo o país), no son datos que identifiquen de manera unitaria e inequívoca a un usuario, garantizando así lo anteriormente comentando. Por otro lado, y respecto a la información relativa al uso del terminal (por ejemplo, las redes sociales), no se recoge ningún tipo de dato sensible que vulnere su privacidad (nombres de usuario, contraseñas, etc.).

A continuación se detalla la estructura, respuesta y contenido de cada uno de los mensajes:

1. **Mensaje de registro (*Tipo 1*):** Mensaje enviado por el cliente tras pulsar el botón "*Registrarme*". Este mensaje puede recibir dos tipos de respuesta:
   - Éxito: El usuario se ha dado de alta en el sistema con éxito. El servidor devuelve al usuario el alias y el identificador único que se le ha asignado.
   - Error: El sistema ha fallado al dar de alta al usuario. El servidor devuelve al usuario un código *404*.

1\n
id_grupo;dispositivo;\n
alias;edad;sexo;país;entorno;estudios

Figura 3.1: Mensaje de *Registro*

2. **Mensaje de datos (*Tipo 2*):** Mensaje enviado por el cliente mientras el usuario tiene la aplicación activa. Este mensaje se envía cada vez que el dispositivo ejecuta alguna de las acciones de las que se desea recopilar información y comentadas en el apartado de *Funcionamiento* de la aplicación.

2\n
id_usuario;dispositivo\n
timestamp;id_evento;id_valor;duración_evento

Figura 3.2: Mensaje de *Datos*

3. **Mensaje de *Login* (*Tipo 3*):** Mensaje enviado por el cliente tras pulsar el botón "*Iniciar sesión*". La respuesta del servidor es similar a la del mensaje de *tipo 1*:
   - Éxito: El usuario ha iniciado sesión con éxito. El servidor devuelve el alias y el identificador único asignado.
   - Error: El sistema ha fallado al iniciar sesión. El servidor devuelve al usuario un código *400*.

3\n
alias;id_usuario;dispositivo

Figura 3.3: Mensaje de *Login*

4. **Mensaje de actualización de datos (*Tipo 4*):** Mensaje enviado por el cliente al servidor cuando se realiza una modificación de los datos inicialmente rellenados en el registro. Permite modificar todos los campos salvo el alias y el dispositivo.

5. **Mensaje de control(*Tipo 5*):** Mensaje enviado por el servidor de control. Permitirá a este producir cambios en el terminal (bloqueo de aplicaciones, información y resúmenes y gráficas diarias de uso, etc.)

4\n
id_usuario;\n
id_grupo;edad;sexo;país;entorno;estudios

Figura 3.4: Mensaje de *Actualización*

En la *Figura 3.5* y *Figura 3.6*, se muestra el flujo temporal de mensajes que se intercambian un dispositivo y el servidor de datos, así como en la *Figura 3.7* el correspondiente al intercambio entre un dispositivo y un servidor de control.

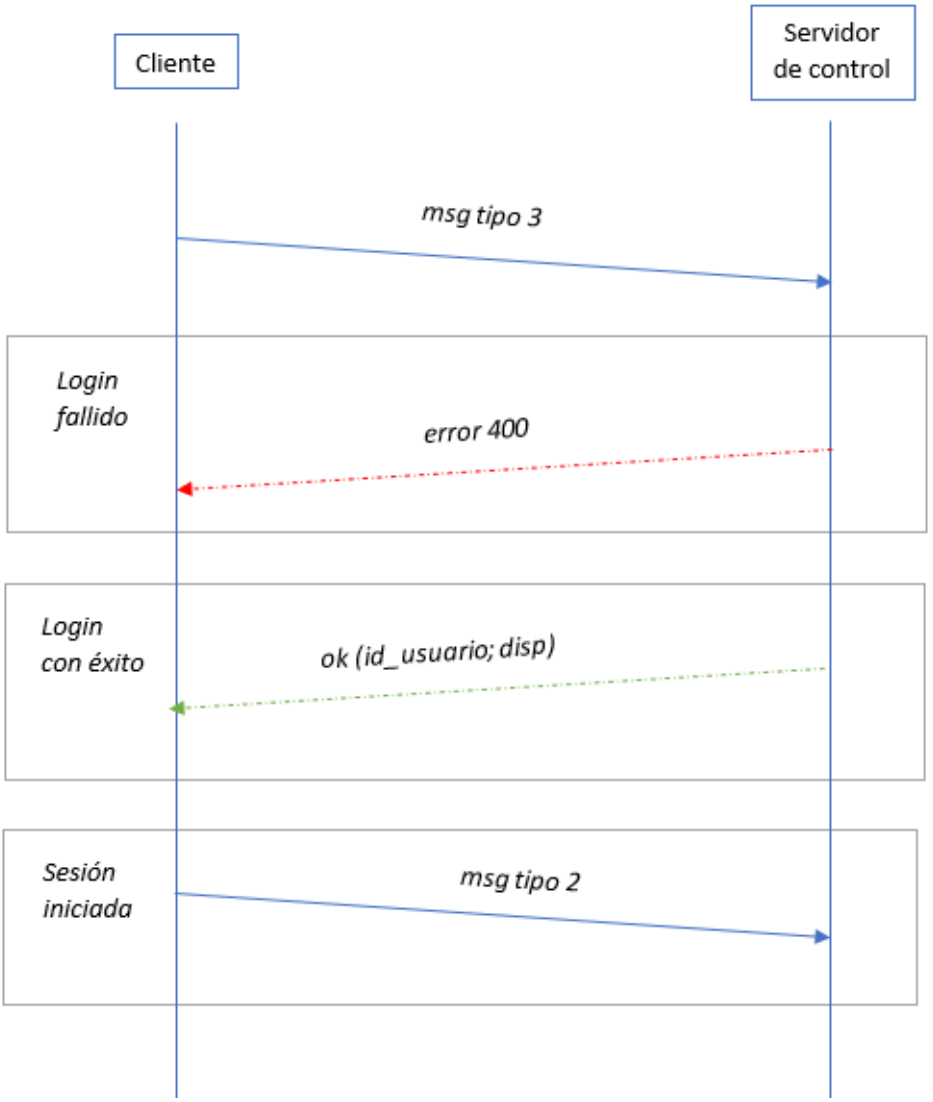


Figura 3.5: Diagrama de flujo temporal de mensajes de *Login* entre un terminal y el servidor de datos

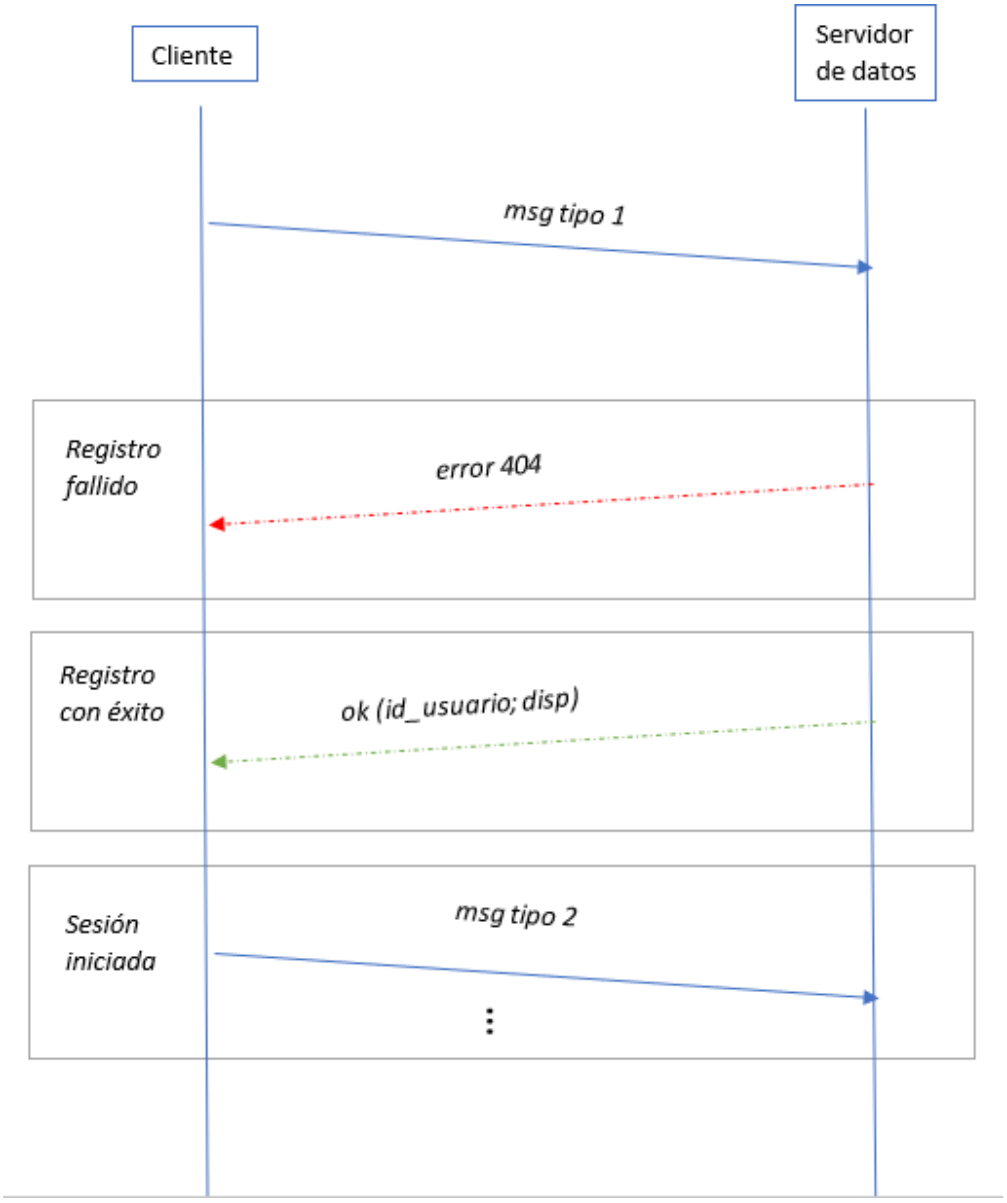


Figura 3.6: Diagrama de flujo temporal de mensajes de *Registro* entre un terminal y el servidor de datos

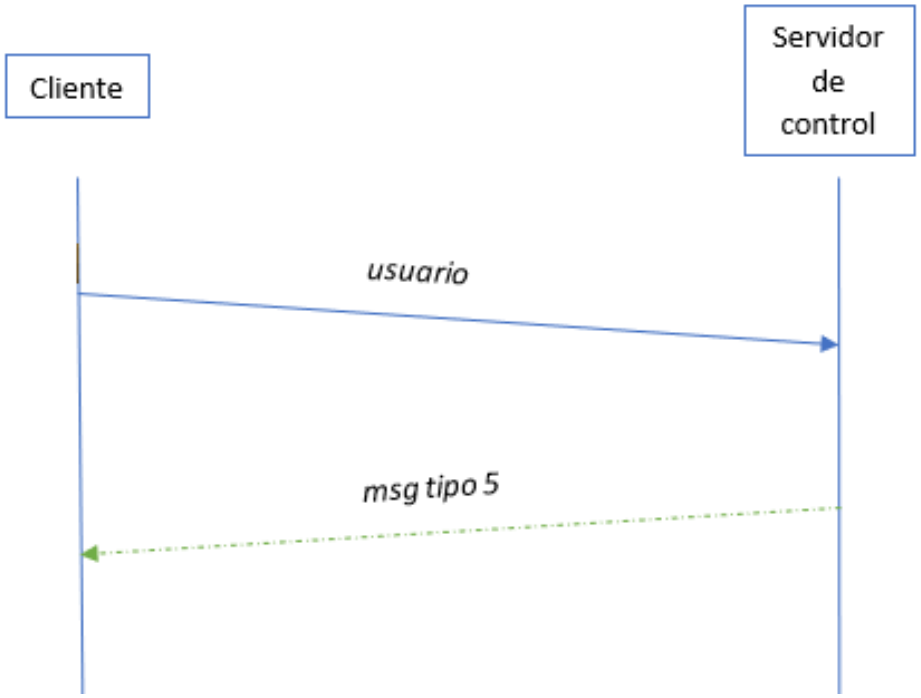


Figura 3.7: Diagrama de flujo temporal de mensajes de *Control* entre un terminal y un servidor de control

3.3. Aplicación Android

3.3.1. Investigación Previa

La necesidad de extraer información de terminales móviles mediante una herramienta fácilmente instalable y para abarcar un número representativo de usuarios posibles, nos lleva al desarrollo de la aplicación, en una primera fase sobre el SO Android por ser éste de amplia implantación.

3.3.2. Requisitos

3.3.2.1. Requisitos funcionales

- **RF1:** La aplicación debe permitir identificar a los usuarios
- **RF2:** La aplicación debe permitir registrar a los usuarios
- **RF3:** La aplicación debe permitir recopilar la siguiente información básica:
  1. Desbloqueo de pantalla
  2. Encendido/apagado de pantalla

3. Uso de RRSS (nombre de la red social y tiempo de uso)
4. Emoticonos empleados

- **RF4:** La aplicación debe permitir la conexión y el intercambio de información con los diferentes servidores tanto de recogida de datos como de control.

3.3.2.2. Restricciones

- La aplicación debe estar realizada en lenguaje para Android (*Java/Kotlin*).

3.3.2.3. Diccionario de datos

- Identificar: el sistema identifica al usuario con un alias y un id numérico único
- Recopilar: Recoger información del dispositivo
- Información: Medidas obtenidas del uso del dispositivo (bloqueo/desbloqueo, uso de emoticonos, encendido/apagado de la pantalla, empleo de RRSS)
- Dispositivo: Terminal con sistema operativo Android (tablet, móvil, chromebook,etc.)
- Comunicarse: Enviar y recibir datos del servidor de datos, así como recibir información de control de otro (u otros) servidor de control.

3.3.3. Diseño

La aplicación está formada por tres *Activities* o pantallas principales. Cada una de ellas se compone de una clase java en la que se define la funcionalidad de la pantalla, y un fichero XML, en el que se define el aspecto de la misma. Además de las pantallas principales, también consta de dos *Fragments* o pantallas secundarias, a las cuales se accederá una vez se haya iniciado sesión o registrado.

- **Login:** Pantalla inicial por defecto de la aplicación. Si un usuario está ya dado de alta y se ha cerrado la sesión, podrá volver a acceder rellenando su alias, un identificador único y pulsando el botón "*Iniciar Sesión*". Si por el contrario el usuario no está registrado, deberá acceder a la pantalla de *Registro* mediante el botón "*Registrarme*".

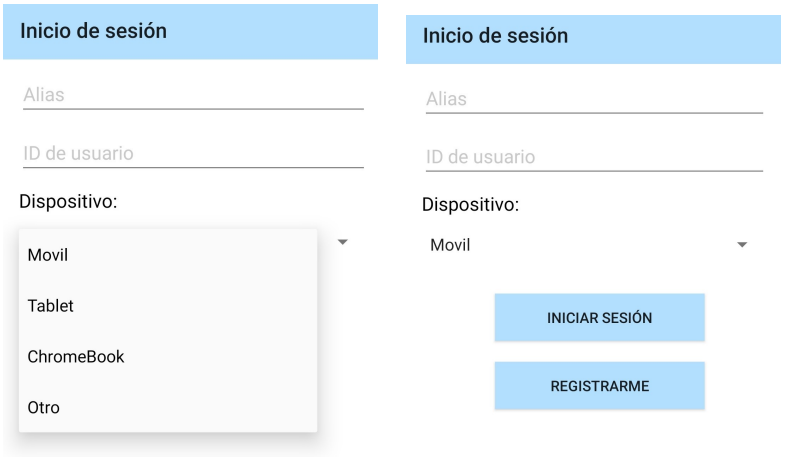


Figura 3.8: Capturas de la pantalla de *Login*

- **Registro:** Pantalla a la que se accede a través del botón "Registrarme" en la de *Login*. Se presenta al usuario un formulario sencillo como el que aparece en la imagen para que rellene los datos correspondientes.

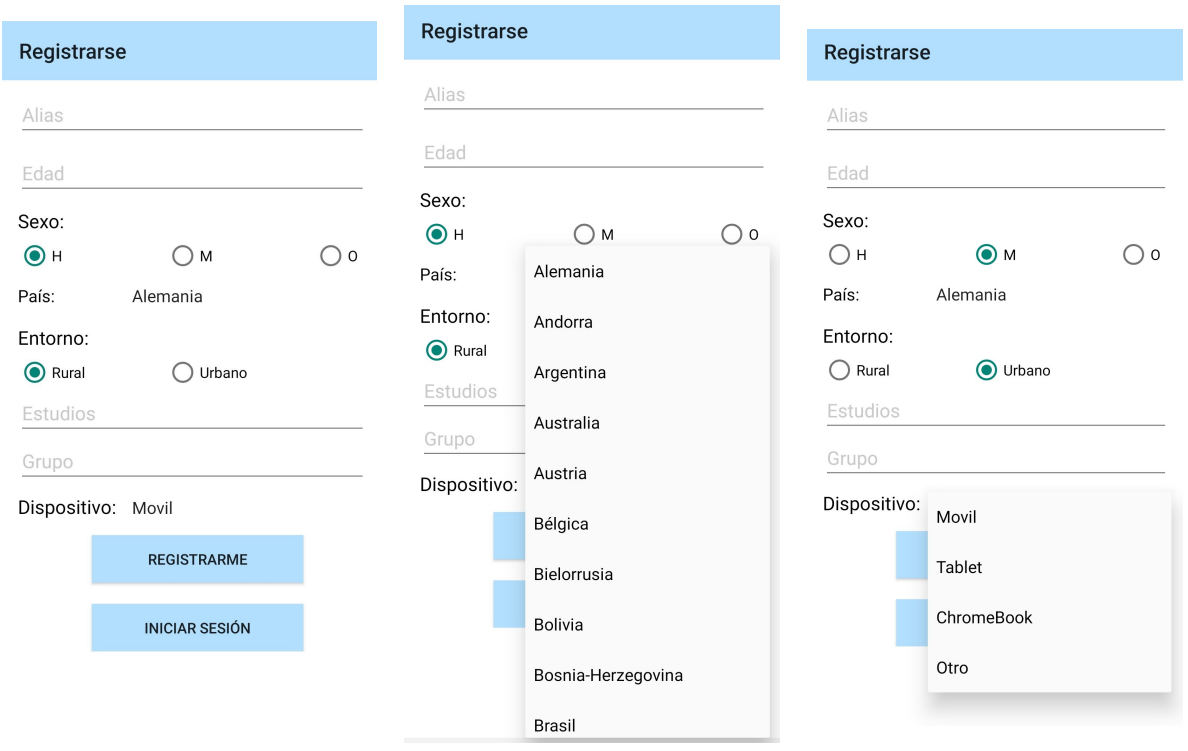


Figura 3.9: Capturas de la pantalla de *Registro*

- **Logged:** Pantalla una vez que el usuario está *logueado* o registrado. En esta pantalla aparecerá un menú lateral que nos permitirá navegar por la aplicación y una página con información recogida del servidor de control relacionada con el usuario.

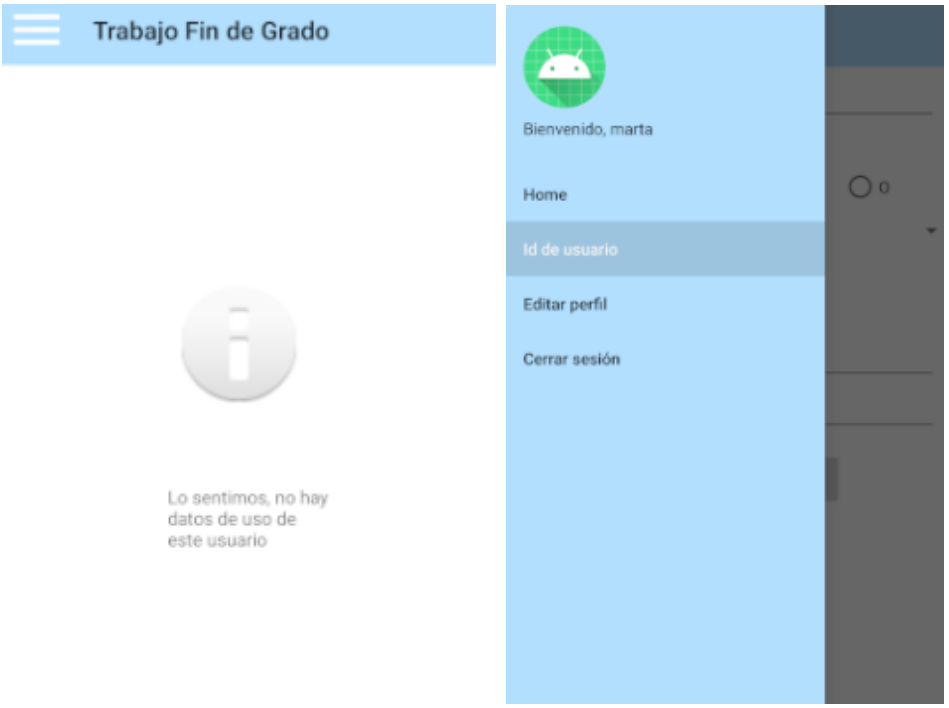


Figura 3.10: Capturas de la pantalla inicial y el menú lateral

Las tres opciones, además de la pantalla principal, a las que nos permite acceder el menú son:

- **Perfil:** Formulario similar al de registro que permite modificar cambios en los datos personales. Constituida por un *Fragment*.

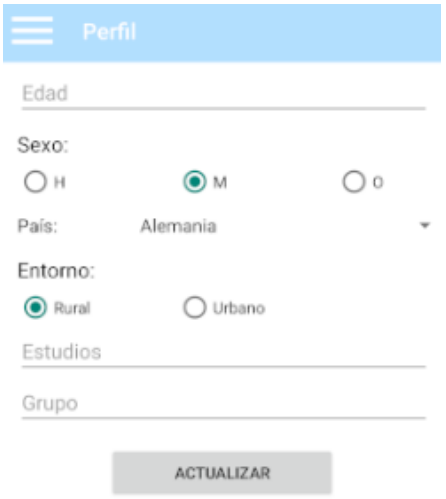


Figura 3.11: Captura de la pantalla *Perfil*

- **Identificador de usuario:** Pop up de información para el usuario que proporciona el identificador único asignado por el servidor y que se emplea para el inicio de sesión.

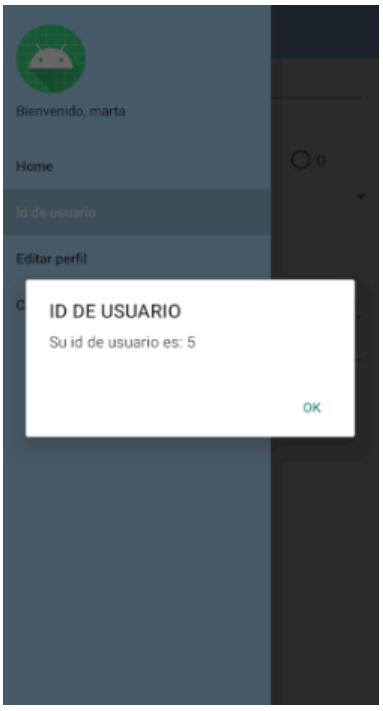


Figura 3.12: Pop up con el identificador de usuario

- **Cerrar sesión:** Vuelta a la pantalla inicial de *Login*.

Además de los elementos visuales, la aplicación consta de un *Servicio* corriendo en segundo plano que permite la recopilación de datos del dispositivo y la elaboración de mensajes para enviar a los servidores. Cuando la aplicación se está ejecutando (ya sea en primer plano o de fondo), aparecerá en la barra de estado del terminal una notificación que indique que se están recogiendo datos, de forma que el usuario sea consciente de ello en todo momento. Junto con el servicio, tenemos también un *Broadcast Receiver* un componente destinado a recibir y responder ante eventos globales del sistema, en nuestro caso bloqueo/desbloqueo y encendido/apagado de la pantalla.

Por otro lado tenemos una clase java que nos permitirá comunicarnos con los servidores y enviar y recibir datos. Dado que la comunicación y la recopilación de datos ha de llevarse a cabo paralelamente, esta clase estará corriendo en un hilo diferente al principal.

3.3.4. Funcionamiento

La funcionalidad de esta aplicación permite que se recoja información concreta de un terminal para posteriormente enviarla a un servidor de datos, así como recibir órdenes del servidor de control en caso de detectar un uso perjudicial del dispositivo por parte del usuario.

Inicialmente, el usuario ha de registrarse o iniciar sesión para poder comenzar el proceso de obtención de datos. A la hora de registrarse, el dispositivo recoge los datos del formulario de la pantalla de "*Registro*" y los guarda en un fichero *registro.txt* dentro del explorador de ficheros del propio dispositivo siguiendo la estructura definida en el protocolo de comunicación como mensaje de *tipo 1*. Por otro lado, si el usuario inicia sesión, de igual manera la información correspondiente se guarda en un fichero *login.txt* que posteriormente es enviado como mensaje de *tipo 3* al mismo servidor de datos.

En ambos formularios, todos los campos serán de obligatorio cumplimiento para poder comprobar el acceso o registrarse. En el caso en el que no se rellene alguno, la aplicación no permitirá continuar y se mostrará un *pop up* que le indique al usuario que rellene todos los campos. Según la respuesta del servidor a estos mensajes (OK o ERROR), se accederá a la aplicación o le aparecerá al usuario un aviso con el error correspondiente.

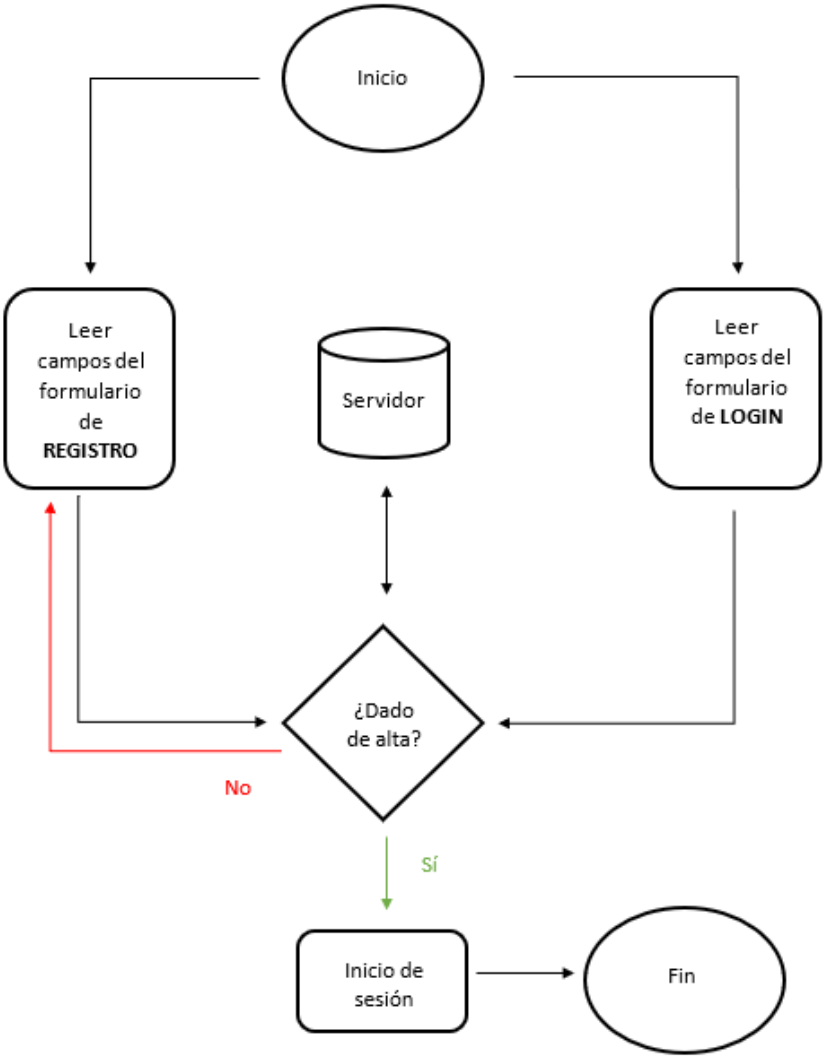


Figura 3.13: Diagrama de flujo de acceso a la aplicación

La *Figura 3.13* muestra un esquema simplificado del flujo que se lleva a cabo para poder iniciar una sesión, ya sea la primera vez, o una vez registrado.

Desde el momento en el que un usuario inicia sesión o se registra en la aplicación, ésta comienza a obtener datos del dispositivo. A cada uno de estos datos se asocia una serie de información que permitirá al experto analizar el uso del dispositivo. Todos los eventos tendrán un identificador de evento y de valor y un *timestamp* (sello temporal), que marcará el momento en el que ha ocurrido. A continuación se muestran los diferentes eventos recogidos y si poseen alguna información adicional.

- Desbloqueo de pantalla
- Encendido/apagado de pantalla
- Redes sociales
    1. Tiempo de uso
- Emoticonos empleados

Cuando en el terminal se sucede un desbloqueo de pantalla o un encendido/apagado de la misma, se manda inmediatamente el mensaje con los datos correspondientes a ese evento al servidor. En el caso del resto de eventos, se envían cada cierto tiempo, el cual viene indicado por el servidor en el mensaje de respuesta.

La página principal de la aplicación mostrará información recibida del servidor de control para postrar pautas de uso del terminal: notificación de alerta, gráficos y tablas resumen de uso desde que se dio de alta, etc.

Como se ha comentado anteriormente, para poder iniciar sesión en el caso en el que ésta se haya cerrado y ya estemos dados de alta, será necesario conocer el identificador único de usuario proporcionado por el servidor de datos. Este identificador estará disponible dentro de la aplicación, para que el usuario pueda consultarlo antes de que se cierre la sesión. En el caso de que no lo conozca, deberá contactar con el administrador (superusuario) para que se lo proporcione.

En el siguiente capítulo hablaremos de los diferentes problemas surgidos en el desarrollo del trabajo y que no han permitido la recopilación de algunos datos del dispositivos por seguridad o funcionalidad de la versión del sistema operativo Android. Es por ello, que ha sido necesario la creación de funciones que emulen la obtención de los siguientes datos:

- Uso de redes sociales
- Emoticonos empleados

Estas funciones generan esta información cada cierto tiempo basándose en probabilidades de uso sobre una lista de elementos de cada tipo de datos.

3.4. Servidor de pruebas

A la hora de elaborar una herramienta que se pueda emplear en un entorno real, inicialmente será necesario simular un entorno de pruebas del mismo, previo a migrar este concepto a un entorno de desarrollo, test y producción con usuarios reales.

Es por ello que en este trabajo se han utilizado los siguientes elementos para poder llevarlo a cabo:

- *VSCode* en SO Windows 10 como servidor de datos de pruebas
- Dispositivo móvil real *Samsung Galaxy S9* con SO Android 10.0
- Dispositivo móvil virtual (emulador) *Google Pixel 3a* con SO Android 11.0

En este apartado desarrollaremos los requisitos que debe tener el servidor de datos de pruebas y su correspondiente funcionamiento.

3.4.0.1. Requisitos funcionales

Al tratarse de un servidor para pruebas, los requisitos establecidos serán los básicos para la comunicación cliente-servidor.

- **RF1:** Conexión con un cliente
- **RF2:** Envío y recepción de datos base definidos en el protocolo de comunicación (*tipo 1, tipo 2, tipo 3 y tipo 4*)

3.4.1. Funcionamiento

Con el fin de probar el correcto funcionamiento de la aplicación sin necesidad de conectarse a servidores con usuarios reales, hemos creado un entorno de pruebas (testing & QA): se ha desarrollado un servidor de pruebas en *Java* para la recogida de datos, que permite la conexión mediante sockets TCP, la recepción y el envío de datos de través de los mismos. Esto no sólo es útil para la programación, sino que podríamos usarlo ya con terminales reales.

Al iniciar la sesión o registrarse un usuario en la aplicación del dispositivo móvil, éste realiza una conexión a la dirección IP y puerto en el cual está escuchando el servidor (en este caso, de pruebas). Una vez establecida la conexión, el servidor comprobará que el mensaje recibido tenga la estructura correcta de acuerdo a la definición de los mensajes de intercambio del protocolo de comunicación escogido entre cliente y servidor. La respuesta del servidor será una simulación de la que se obtendría por parte del servidor remoto, contemplando tanto la respuesta de éxito como los posibles errores generados (404/400 Error).

En el caso de este servidor no va a ser necesario comparar la información de usuarios ya registrados en la base de datos, puesto que lo que se pretende es comprobar cómo responde la aplicación ante los posibles mensajes recibidos por el servidor de datos real.

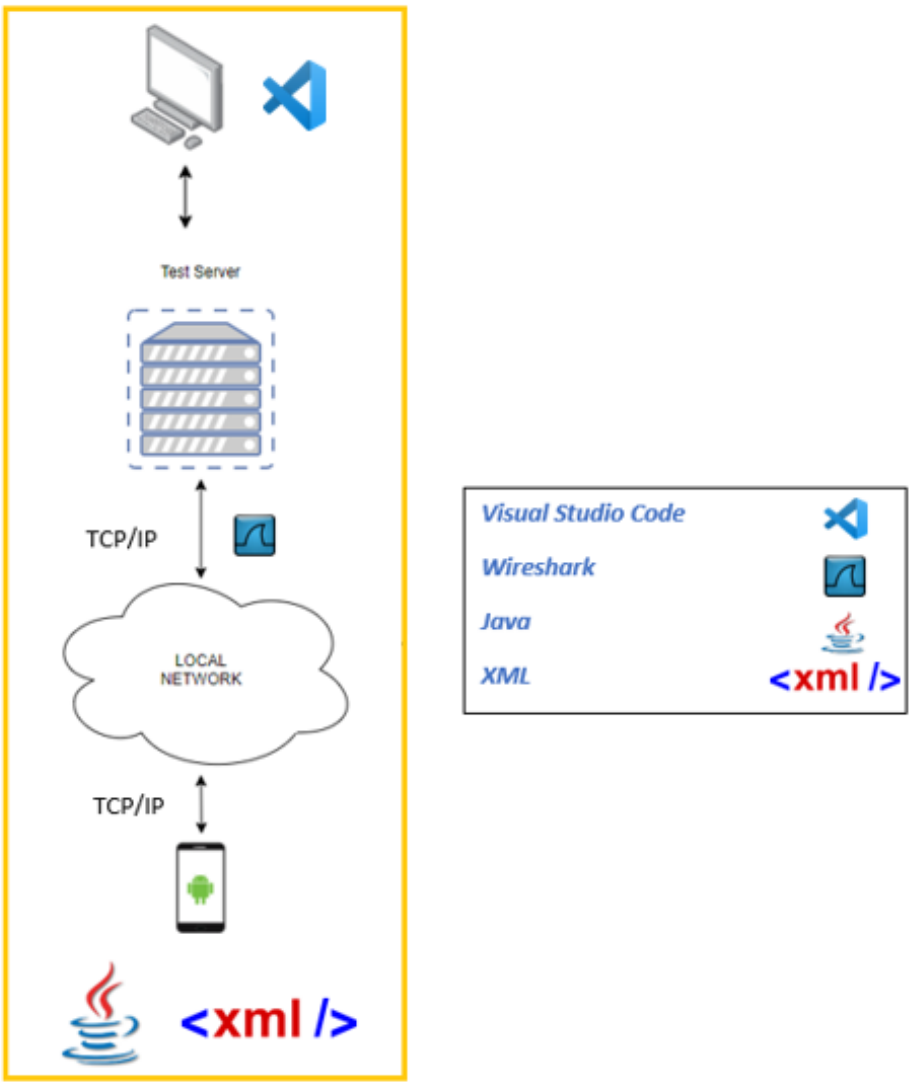


Figura 3.14: Esquema representativo de la arquitectura de pruebas y herramientas utilizadas

3.5. Servidores de desarrollo beta

Para poder diferenciar distintos aspectos de la comunicación de los terminales con un servidor, se planteó el uso de dos tipos de servidores diferentes:

- Servidor de datos: Recopila datos recogidos por el dispositivo para almacenarlos.
- Servidor/es de control: Permiten envío de información de control al dispositivo para controlar su funcionamiento el función del uso del mismo por parte del usuario.

En este apartado se detallarán los requisitos y funcionalidades de los diferentes servidores desde el punto de vista del cliente.

3.5.1. Servidor de datos

3.5.1.1. Requisitos funcionales

- **RF1:** Conexión con un cliente
- **RF2:** El servidor de datos llevará a cabo todo lo relacionado con el almacenamiento de datos recogidos por el terminal. En el apartado *3.2. Protocolo de comunicación*, las *Figuras 3.5, 3.6 y 3.7* muestran diagramas temporales de los mensajes intercambiados

3.5.1.2. Funcionamiento

Como se ha comentado en apartados anteriores, este servidor permitirá la recogida y almacenamiento de datos enviados por el dispositivo, que en principio y por simplificar el modelo vamos a considerar único. Con lo que respecta al protocolo de comunicación los mensajes *tipo 1, tipo 2, tipo 3 y tipo 4* se intercambiarán con este servidor.

La comunicación entre ambos elementos será bidireccional, de manera que entre servidor y dispositivo no se envíen solo la información a almacenar recopilada por el dispositivo al servidor, sino también la respuesta de éste con información necesaria en el terminal (como el identificador de usuario) o la confirmación o error en la llegada de los datos. Aunque en un futuro, conforme el modelo vaya evolucionando, se puedan plantear la existencia de diferentes servidores de datos, al igual que de control, en este proyecto se va a emplear inicialmente un único servidor de datos.

3.5.2. Servidor de control

3.5.2.1. Requisitos funcionales

- **RF1:** Conexión con un cliente
- **RF2:** Envío y recepción de datos de acuerdo con el protocolo de comunicación establecido (mensaje *tipo 5*)

3.5.2.2. Funcionamiento

Como se ha comentado en apartados anteriores, este sistema se plantea también como una herramienta que facilite la actuación al superusuario (experto) actuar sobre el dispositivo en el caso de que se detecte un comportamiento que pueda suponer un riesgo para el usuario.

En el protocolo de comunicación definido en el apartado 3.2, se plantea un mensaje de control (*tipo 5*), que permitirá realizar ciertas acciones sobre el dispositivo (bloqueo de determinadas aplicaciones, información, resúmenes y gráficas mostradas al usuario sobre el uso que ha tenido desde el inicio de sesión, etc.) Cabe destacar que esta actuación por parte de los expertos estará supeditada a garantizar la seguridad y anonimidad del usuario, así

como al conocimiento del proceso, consentimiento del protocolo de emergencia, y expresa autorización del mismo.

La siguiente figura muestra el flujo de recogida de datos por parte del servidor de datos (morado) y la alarma con la consecuente actuación (rojo) de uno de los servidores de control.

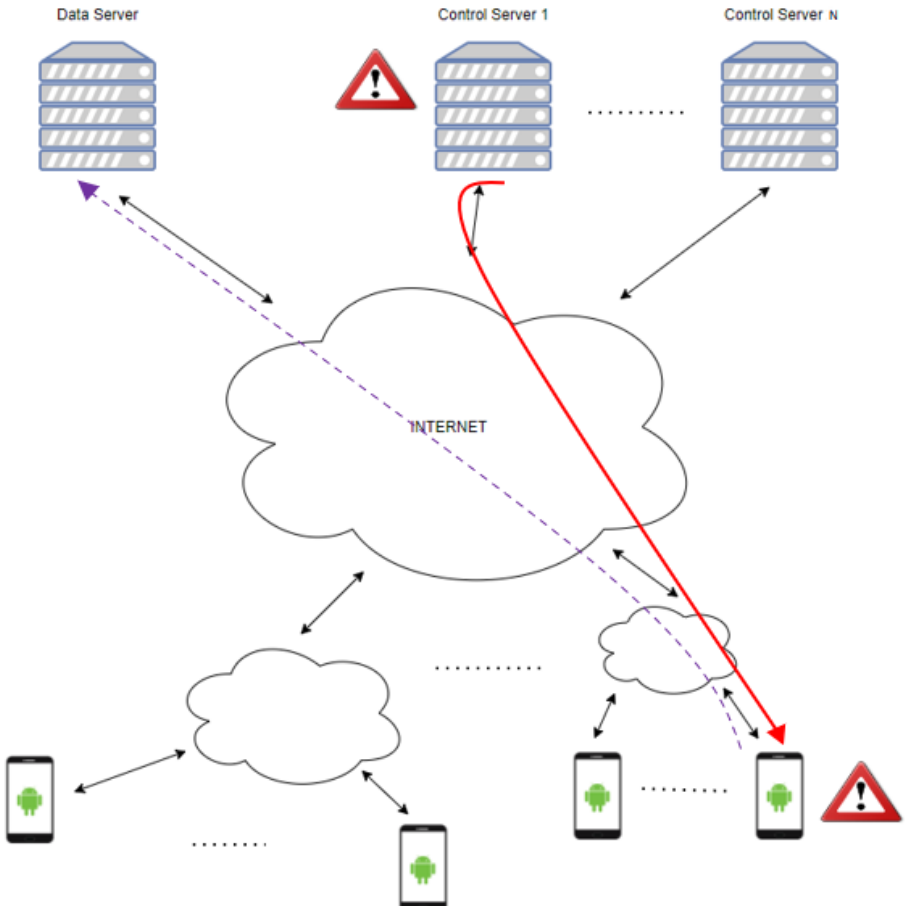


Figura 3.15: Alerta de riesgo de un usuario

Capítulo 4

Pruebas

4.1. Pruebas durante el desarrollo

Durante el desarrollo de la aplicación se han llevado a cabo pruebas para verificar el correcto funcionamiento de la misma. Para ello, se han empleado los siguientes elementos:

- **Dispositivo virtual:** Emulador de Pixel 3a con SO Android 11.0 (API 30) en Android Studio.
- **Dispositivo real:** Samsung Galaxy S9 con SO Android 10.0 (API 29)
- **Servidor de pruebas:** Virtual Studio Code
- **Servidor de producción:** gtc1pc1.cps.unizar.es
- **Analizador de tráfico:** Wireshark

4.1.1. Limitaciones surgidas

Al realizar las pruebas empleando los elementos mencionados anteriormente nos encontramos con problemas de diferentes tipos:

- Recursos
- Referentes a la versión del SO

  Funcionalidad

  Seguridad
- Referentes a las conexiones

4.1.1.1. Limitaciones con los recursos

Como se ha comentado, el desarrollo de esta herramienta se ha llevado a cabo con el programa Android Studio, que incluye no solo la programación de la propia aplicación, sino también un entorno de virtualización de dispositivos (emuladores) con diferentes versiones Android para hacer pruebas. Este entorno de desarrollo consume bastantes recursos del ordenador donde se esté ejecutando, por lo que, en este caso, ha supuesto una ralentización en los momentos de ejecución y pruebas.

4.1.1.2. Restricciones del Sistema Operativo: Funcionalidad

Las nuevas versiones de los lenguajes de programación traen consigo mejoras como nuevas funcionalidades u optimizaciones de las mismas. De igual manera, en las actualizaciones de versión de los sistemas operativos nos encontramos cambios en la apariencia pero también de funcionalidad, otorgando al usuario nuevas herramientas dentro del mismo dispositivo.

En nuestra aplicación, algunas funciones necesarias para la recogida de datos requieren que el terminal tenga una versión de la API igual o superior a la 26, es decir, a partir de Android 7.0 (Android Nougat).

4.1.1.3. Restricciones del Sistema Operativo: Seguridad

Por otro lado, además de nuevas funcionalidades se producen cambios con respecto a la seguridad del sistema, elaborando parches para solventar errores o simplemente elaborando nuevas medidas que permitan que el uso del dispositivo sea más seguro para el usuario.

Una de las medidas de seguridad en los sistemas operativos recae en el uso del método de entrada de texto (teclado, pistola de lectura...). Para garantizar la privacidad del usuario, no se permite la lectura de teclado desde aplicaciones externas a la que se ha abierto. Esto, podría suponer una grave vulnerabilidad, dando, por ejemplo, opción a obtener datos críticos como contraseñas o datos sensibles de los usuarios.

Además de la entrada de texto, y en posible combinación con lo anterior, otro de los problemas que nos hemos encontrado en este trabajo es la obtención de la lista de procesos (aplicaciones) que se ejecutan en el dispositivo. Conocer la aplicación que se está ejecutando en primer plano junto con la lectura de datos introducidos por el usuario podría suponer un potencial riesgo para los usuarios, pudiendo ocasionar, por ejemplo, el hackeo por parte de terceros de cuentas bancarias, redes sociales, etc.

4.1.1.4. Limitaciones referentes a las conexiones

Otro de los problemas surgidos en este trabajo está relacionado con las conexiones entre cliente y servidor de pruebas mediante los sockets TCP. En ocasiones, realizando pruebas del desarrollo, han saltado excepciones de la programación que han impedido el funcionamiento de la misma y, por consiguiente, el correcto cierre de los sockets tanto en el dispositivo virtual (emulador) como en el real, y en el SO Windows de los ordenadores donde se ha llevado a cabo el trabajo. Es por ello, que al reiniciar el programa para continuar las pruebas, en ocasiones ha sido necesario cambiar de puerto o eliminar el proceso que se ha quedado corriendo en el mismo.

4.2. Pruebas de análisis

Una vez finalizada la aplicación, se han llevado a cabo las pruebas para comprobar el correcto funcionamiento de la misma junto con los servidores de producción. Para ello hemos

empleado el analizador de tráfico *Wireshark*, que nos ha permitido ver la correcta comunicación entre los elementos del sistema. Las siguientes capturas corresponden a la prueba de comunicación entre el emulador antes mencionado y los servidores de desarrollo beta.

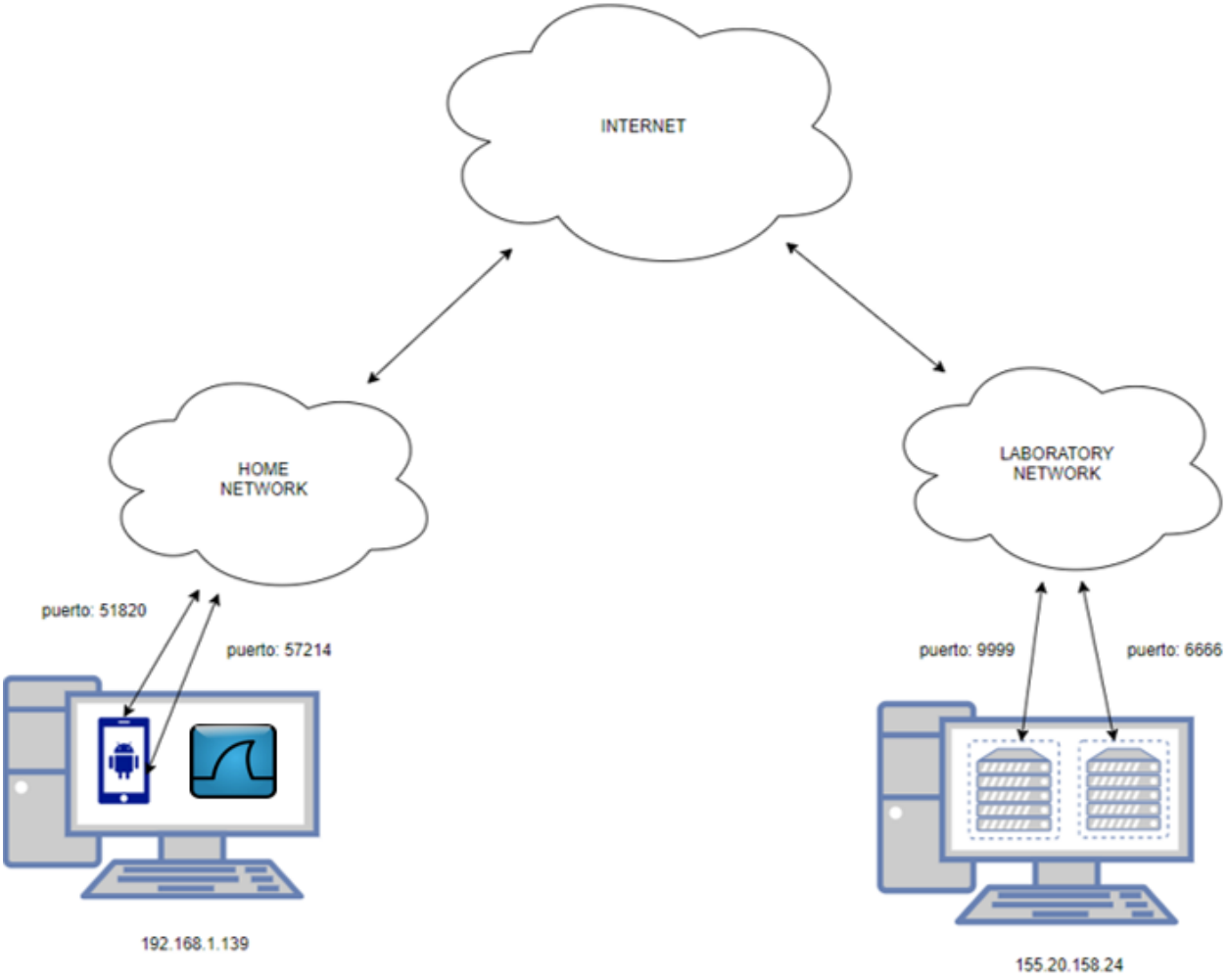
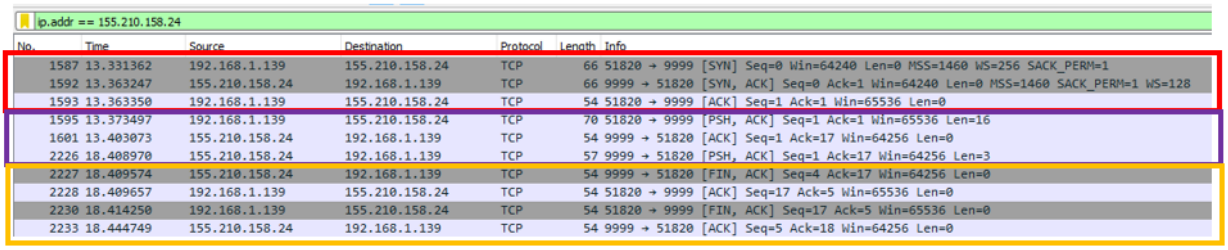


Figura 4.1: Arquitectura de pruebas

El protocolo TCP es un protocolo orientado a conexión, por lo que las conexiones entre elementos con este protocolo se componen de tres etapas:

1. Establecimiento de conexión o *3-way handshake*
2. Transferencia de datos
3. Fin de la conexión

En las *Figuras 4.2* y , se muestra claramente cada una de las etapas correspondientes a la conexión TCP.



ip.addr == 155.210.158.24

No.	Time	Source	Destination	Protocol	Length	Info
1587	13.331362	192.168.1.139	155.210.158.24	TCP	66	51820 → 9999 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
1592	13.363247	155.210.158.24	192.168.1.139	TCP	66	9999 → 51820 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM=1 WS=128
1593	13.363350	192.168.1.139	155.210.158.24	TCP	54	51820 → 9999 [ACK] Seq=1 Ack=1 Win=65536 Len=0
1595	13.373497	192.168.1.139	155.210.158.24	TCP	70	51820 → 9999 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=16
1601	13.403073	155.210.158.24	192.168.1.139	TCP	54	9999 → 51820 [ACK] Seq=1 Ack=17 Win=64256 Len=0
2226	18.408970	155.210.158.24	192.168.1.139	TCP	57	9999 → 51820 [PSH, ACK] Seq=1 Ack=17 Win=64256 Len=3
2227	18.409574	155.210.158.24	192.168.1.139	TCP	54	9999 → 51820 [FIN, ACK] Seq=4 Ack=17 Win=64256 Len=0
2228	18.409657	192.168.1.139	155.210.158.24	TCP	54	51820 → 9999 [ACK] Seq=17 Ack=5 Win=65536 Len=0
2230	18.414250	192.168.1.139	155.210.158.24	TCP	54	51820 → 9999 [FIN, ACK] Seq=17 Ack=5 Win=65536 Len=0
2233	18.444749	155.210.158.24	192.168.1.139	TCP	54	9999 → 51820 [ACK] Seq=5 Ack=18 Win=64256 Len=0

Figura 4.2: Conexión cliente-servidor de datos

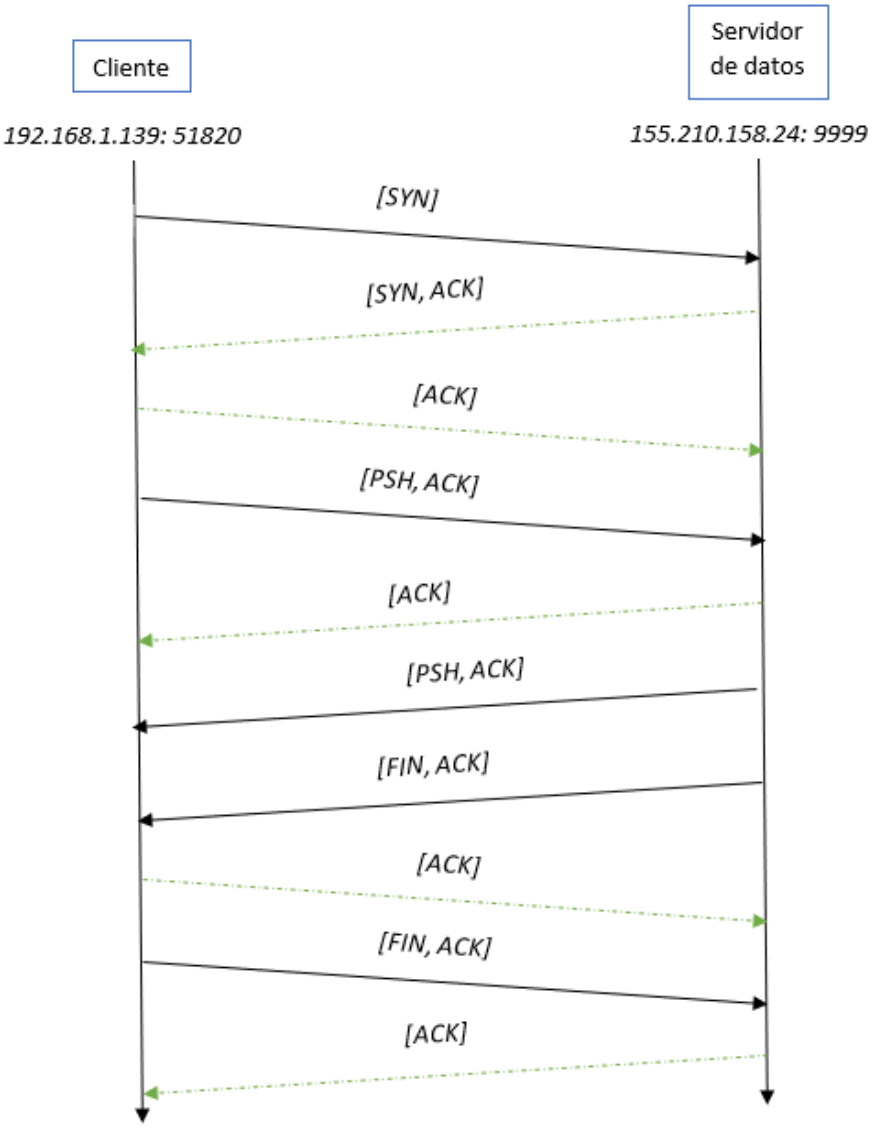


Figura 4.3: Diagrama de flujo temporal de pruebas cliente-servidor de datos

En el recuadro de color rojo, los flags *[SYN], [SYN, ACK], [ACK]* nos indican el establecimiento de la conexión entre el cliente (192.168.1.139: 51820) y el servidor (155.210.158.24:9999). Por otro lado, el recuadro naranja nos muestra con la secuencia *[FIN, ACK], [FIN, ACK]* el final de la conexión.

Respecto a la transferencia de datos, vamos a comprobar la comunicación de acuerdo con el protocolo previamente definido. Para que el análisis sea más sencillo, hemos añadido los filtros ***tcp.port == 9999*** y ***tcp.flags.push == 1*** a los paquetes capturados, de manera que podamos ver los correspondientes a la comunicación con el servidor de datos (puerto destino 9999) y aquellos paquetes que envíen datos recogidos por el dispositivo (*[PSH, ACK]*).

El primer mensaje que vamos a enviar es el de *Login*, conociendo previamente que el usuario "*marta*" está dado de alta en el sistema con el identificador 1. Sin embargo, y debido a un error, el usuario que intenta iniciar sesión contiene un fallo, siendo "*martu*" el alias que se envía. En la parte inferior de la imagen se puede observar el mensaje completo enviado al servidor, correspondiente al *tipo 3* del protocolo.

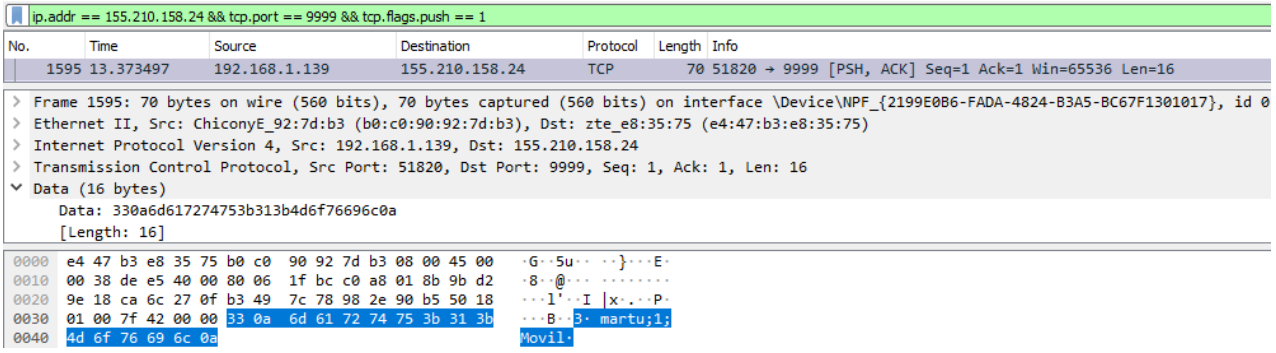


Figura 4.4: Mensaje de *tipo 3*

Al no encontrarse este usuario registrado en el sistema, el servidor nos devuelve un mensaje de error con el código *400*.

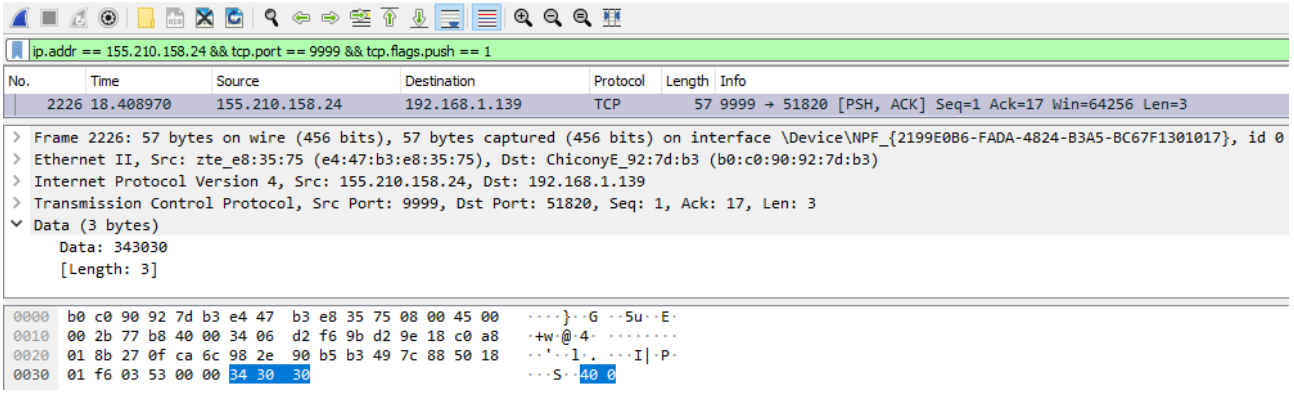


Figura 4.5: Error en el *Login*

Esta vez vamos a escribir el alias correctamente y comprobamos que el servidor nos envía una respuesta correspondiente a inicio de sesión con éxito.

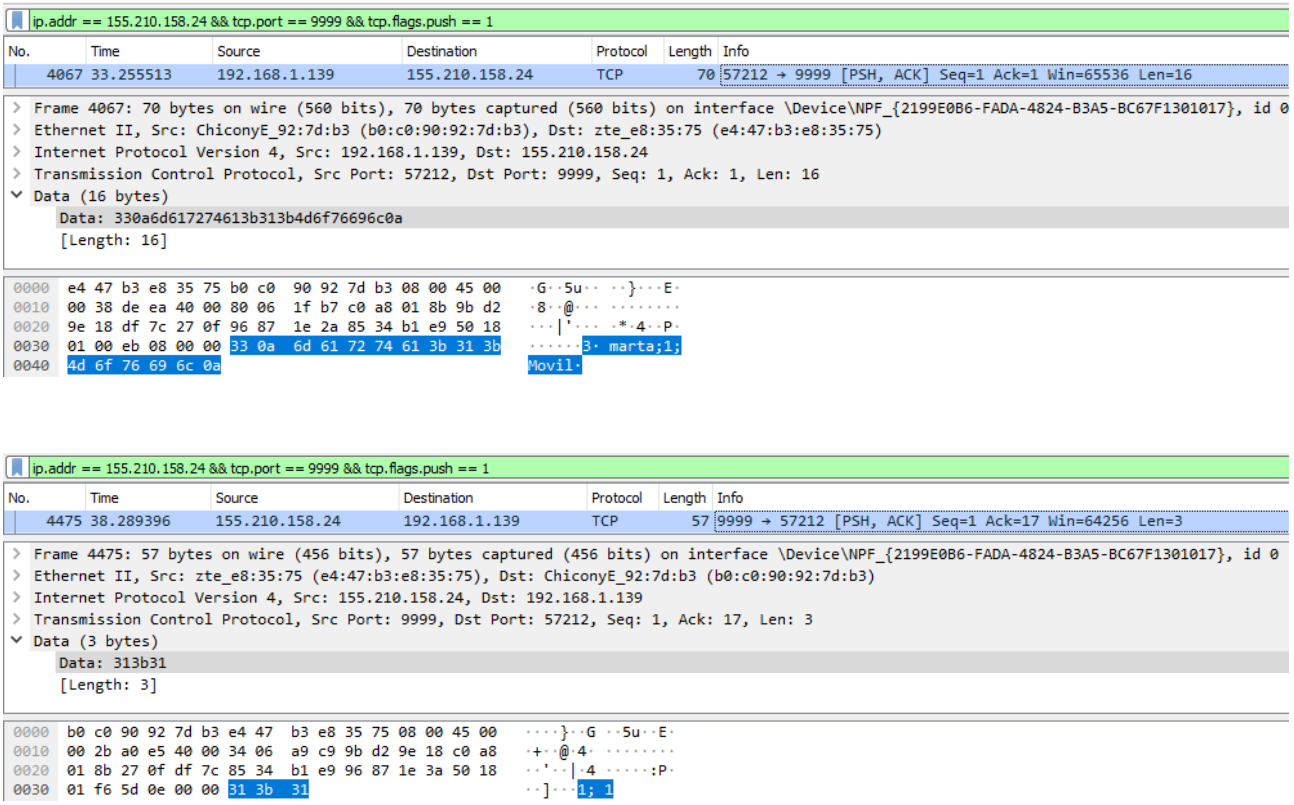


Figura 4.6: Inicio de sesión correcto

Al igual que con el inicio de sesión, vamos a ver el comportamiento de los mensajes de *Registro*.

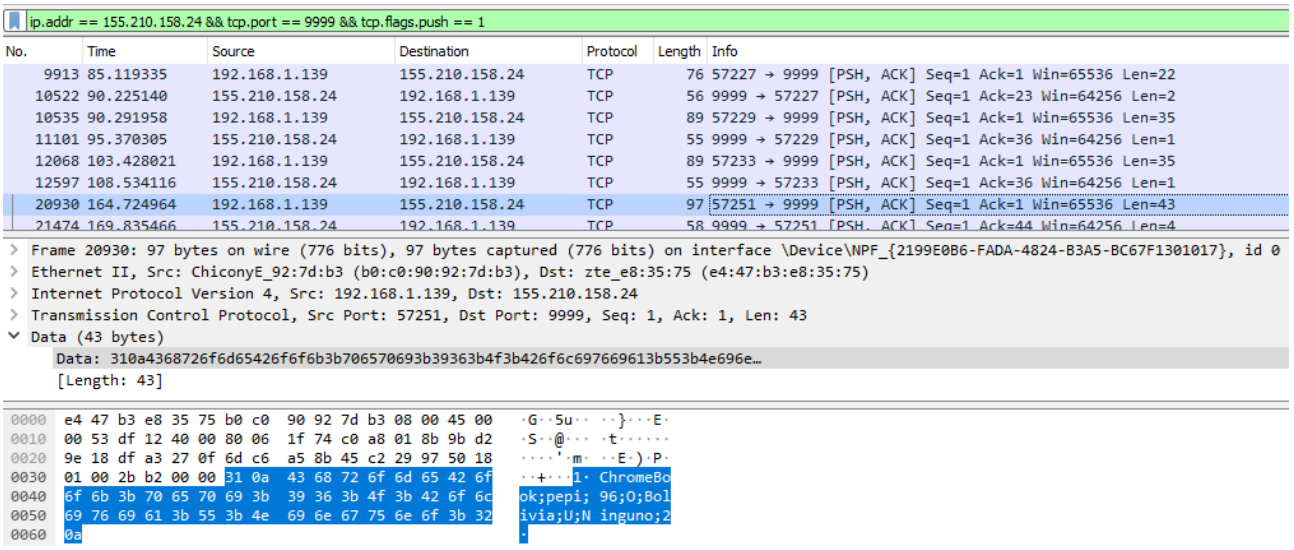


Figura 4.7: Mensaje *tipo 1* del dispositivo

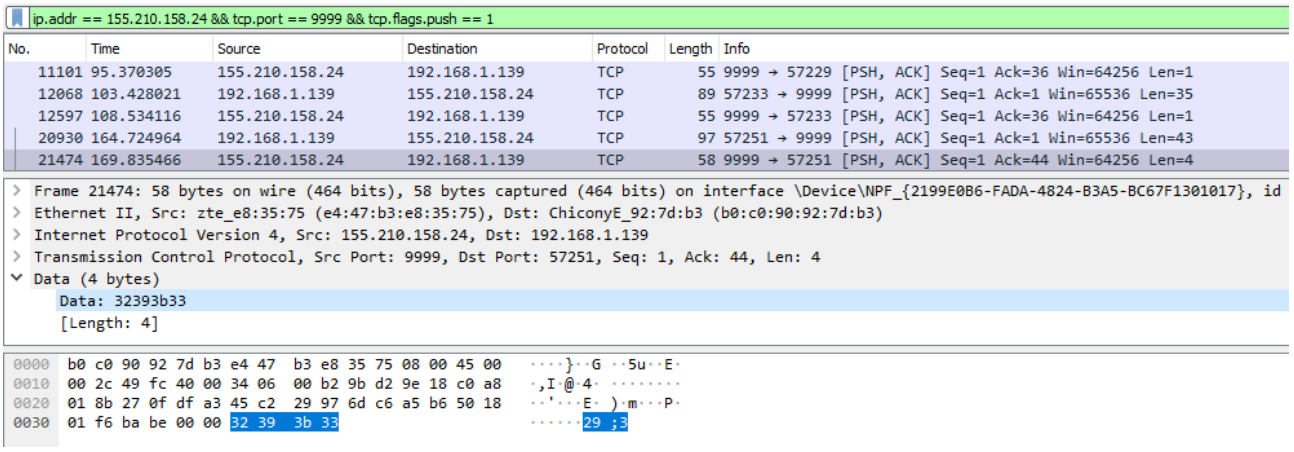


Figura 4.8: Registro con éxito en el servidor

Una vez dentro de la aplicación, el dispositivo enviará datos de uso al servidor (mensaje *tipo 2*), y obtendrá como respuesta el tiempo de actualización del mismo, es decir, la frecuencia a la que es capaz de procesar los datos enviados.

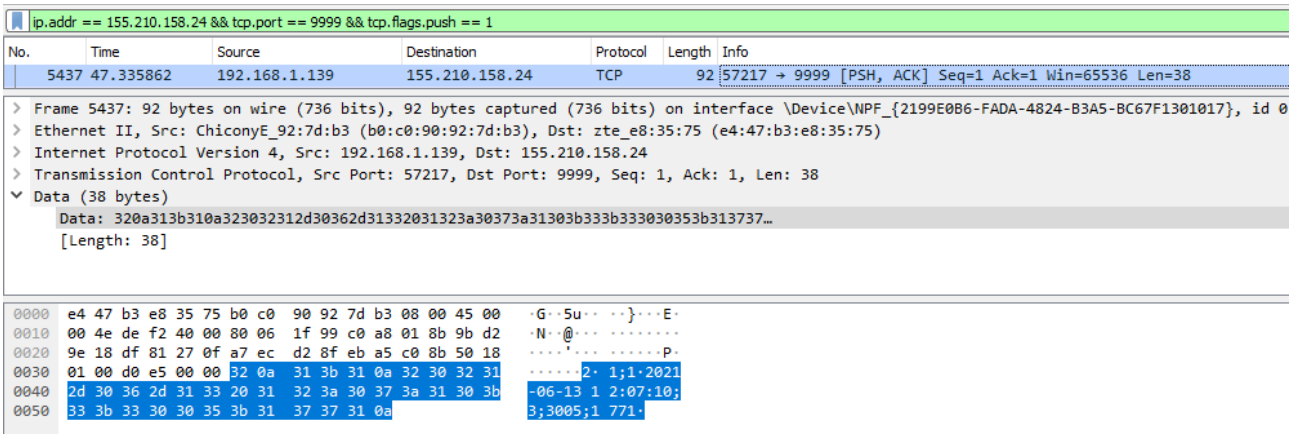


Figura 4.9: Mensaje *tipo 2* del dispositivo

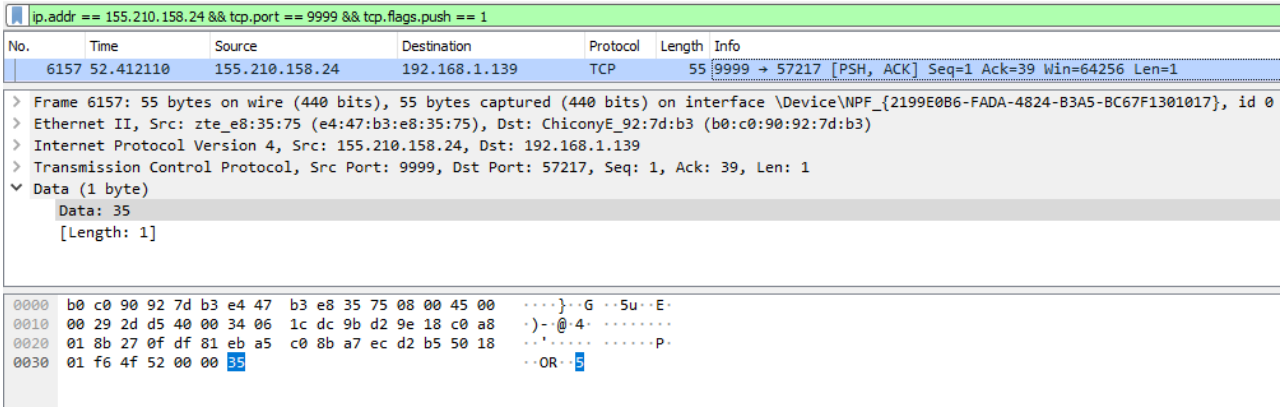


Figura 4.10: Tiempo de actualización en del servidor

Junto con los datos de uso del propio dispositivo, otro de los mensajes que puede recibir el servidor es el de actualización de la información del usuario de la sesión.

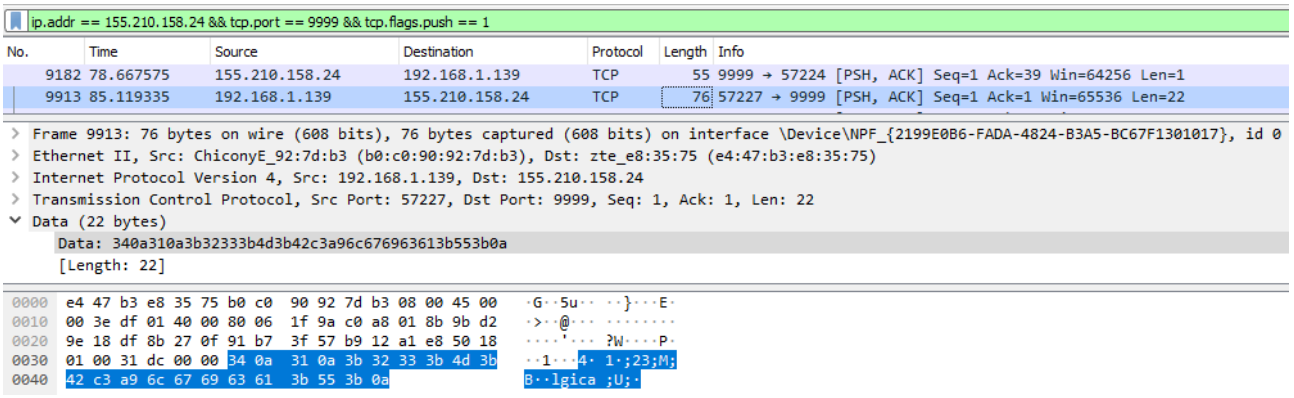


Figura 4.11: Mensaje *tipo 4* del dispositivo

Además de la conexión con el servidor de datos, también se ha comprobado con el de control. Al igual que en el caso anterior, filtraremos por el puerto del servidor de control: ***tcp.port == 6666*** y situado en la misma dirección IP.

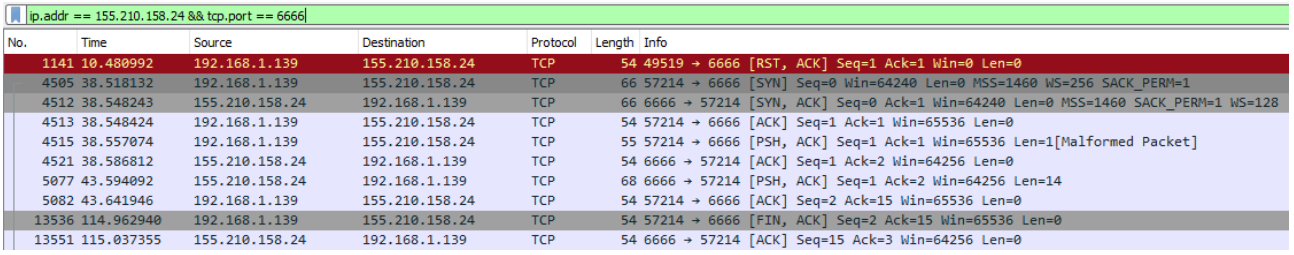


Figura 4.12: Conexión cliente-servidor de control

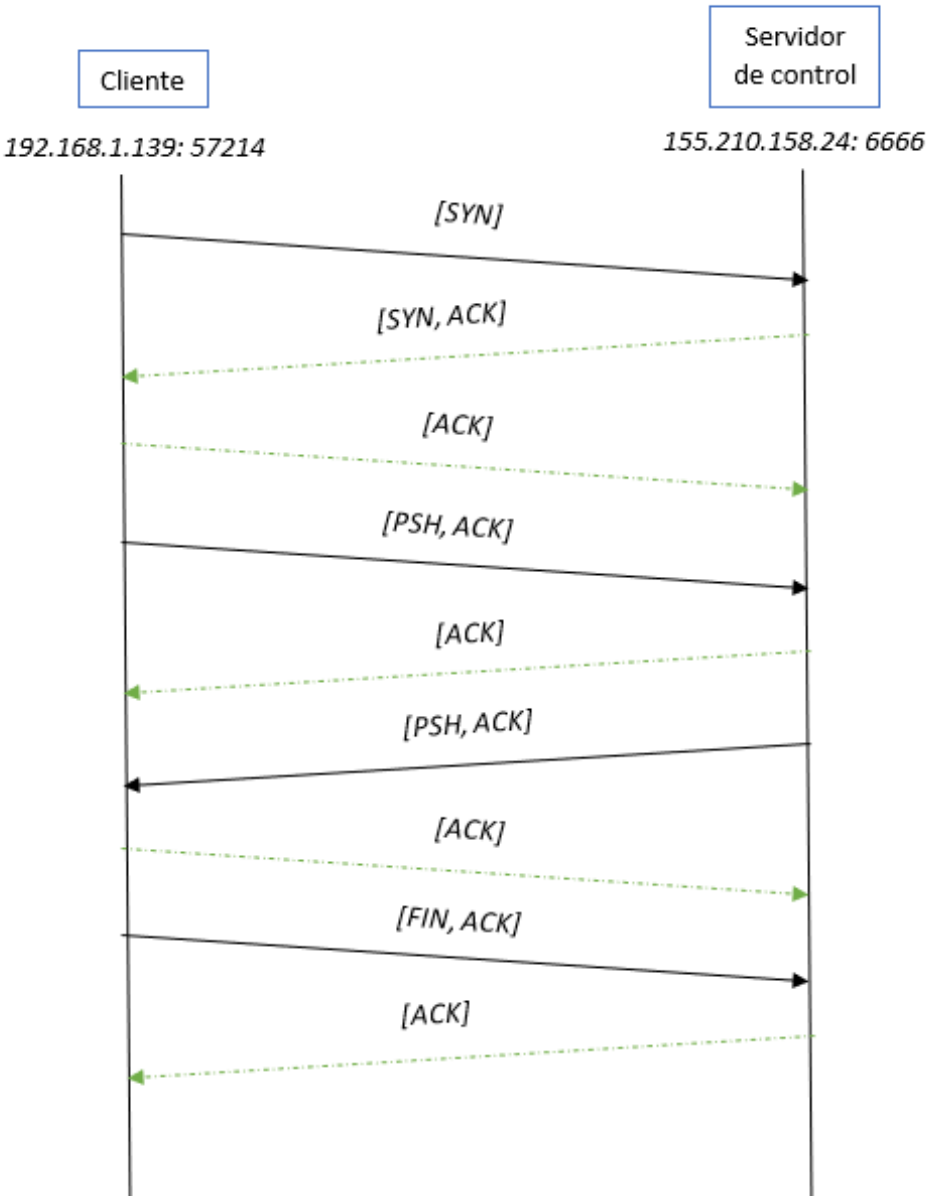


Figura 4.13: Diagrama de flujo temporal de pruebas cliente-servidor de control

Los mensajes de *tipo 5*, que aunque de cara a futuro permitirán realizar determinadas acciones de control sobre el dispositivo (bloqueo de determinadas aplicaciones, envío de gráficas, etc.), en este prototipo nos servirán para recibir información de dicho servidor. El cliente le enviará un mensaje con su identificador de usuario para poder identificarlo de manera única.

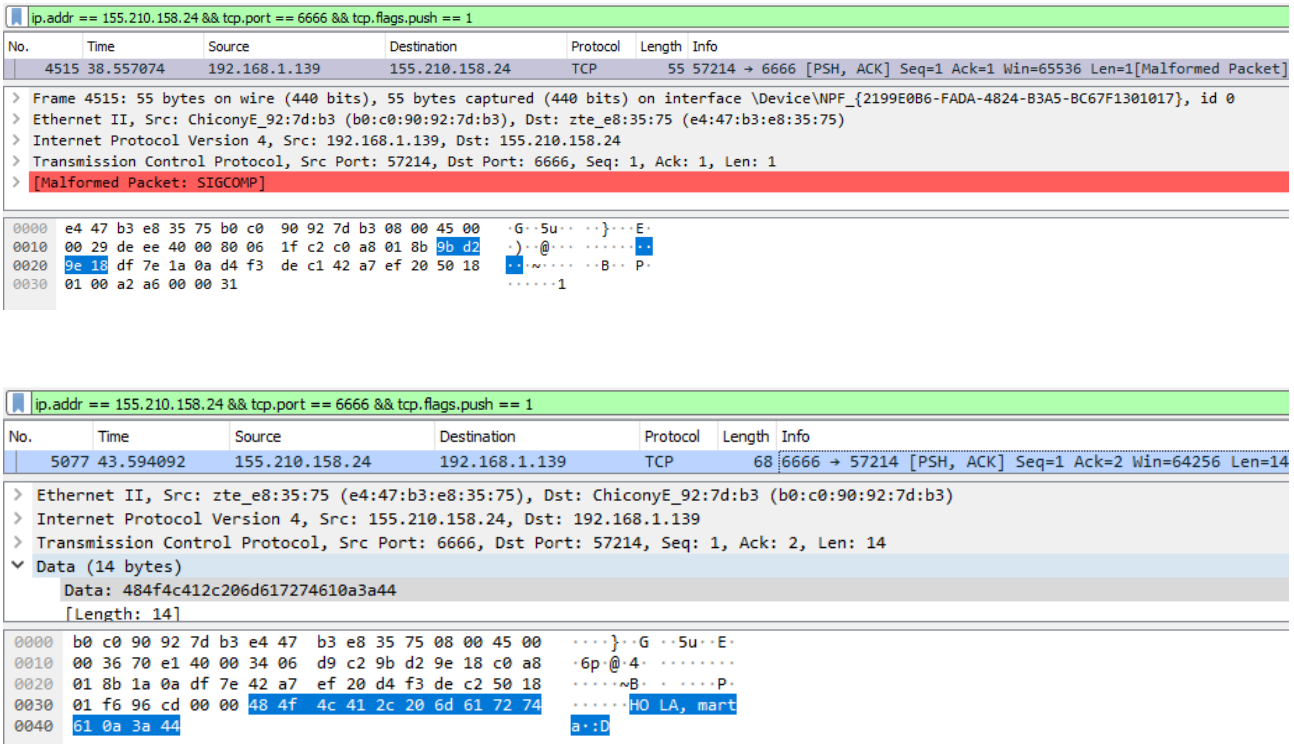


Figura 4.14: Intercambio de mensajes entre servidor de control y dispositivo

Capítulo 5

Conclusiones y líneas futuras

5.1. Conclusiones

Este proyecto se ha realizado dentro de un marco de investigación colaborativo junto al grupo EDUCAVIVA de la Facultad de Educación de la Universidad de Zaragoza, con la finalidad de otorgarles una herramienta que mejore el estudio, desde la perspectiva educativa, de la QoE, dentro de la QoX, relacionada con el uso de las nuevas tecnologías.

Con este trabajo se han conseguido los siguientes objetivos:

1. Recopilación de datos de un dispositivo Android (tablet, móvil, chromebook, etc) compatible con la versión Android Nougat (7.0) y/o superiores.
2. Conexión simultánea de un terminal a diferentes servidores, intercambiando información con ellos de manera concurrente a la recopilación de datos, así como el tratamiento de los datos recibidos.
3. Elaboración de un protocolo de comunicación básico, con posibilidad de escalabilidad, basado en la arquitectura básica cliente-servidor.
4. Emulación de tareas que no han podido ser implementadas por limitaciones de la tecnología y hacen transparente su función.

5.2. Líneas futuras

Una vez desarrollado un prototipo base de la herramienta, destacan como líneas futuras:

- Evolucionar el sistema de comunicación con los servidores de forma que se pueda ampliar la información intercambiada, es decir, mejorar la escalabilidad del sistema respecto a la comunicación.
- Desarrollar en su totalidad el sistema de comunicación definido con el servidor de control, permitiendo a los expertos actuar, de manera controlada y asegurando la privacidad y seguridad del usuario, sobre el dispositivo.
- Optimizar la aplicación ya creada, respecto a código y funcionalidad, de manera que mantenga el equilibrio entre seguridad y recopilación de datos. Realización de tareas que no han sido posibles de implementar mediante un análisis más profundo de las funcionalidades de la tecnología.

En el caso de querer implantarse en un entorno real, la herramienta deberá ser optimizada, revisada y verificada en su totalidad por expertos. Además, se deberá realizar un formulario de aceptación escrita para la recogida de datos firmado por el usuario, garantizando así su pleno conocimiento, voluntad y conformidad a ello.

Bibliografía

[1] Documentación desarrollo *Android*:
`https://developer.android.com/guide`

[2] *Java*:
`https://www.oracle.com/java/`

[3] *Wireshark*:
`https://www.wireshark.org/`

[4] *Visual Studio Code*:
`https://code.visualstudio.com/`

[5] Comunicación TCP por sockets:
Apuntes de la asignatura "*Programación de redes y servicios*"

Acrónimos

TFG - Trabajo Fin de Grado

QoX - Quality of X (Calidad de X)

QoE - Quality of Experience (Calidad de Experiencia)

QoS - Quality of Service (Calidad de Servicio)

VSCode - Visual Studio Code

TCP - Transmission Control Protocol (Protocolo de Control de Transmisión)

IP - Internet Protocol (Protocolo de Internet)

SO - Sistema Operativo

API - Application Programming Interface (Interfaz de Programación de Aplicaciones)

RRSS - Redes Sociales

ANEXOS

Anexo 1. Instalación y configuración de herramientas para el desarrollo

Para la realización de este trabajo ha sido necesario la instalación de las siguientes herramientas:

- Visual Studio Code
- Android Studio
- Java
- Wireshark

Visual Studio Code

En este proyecto empleamos este editor de código multiplataforma para la implementación del servidor de pruebas. Esta herramienta permite el uso de un terminal (o varios a la vez) integrado en su interior, lo que supone una ventaja para la ejecución y depuración de código. Este programa está disponible para Windows, macOS y Linux. La página oficial permite descargar el instalador, al que posteriormente se le pueden añadir extensiones para los estilos o la optimización de código, así como la integración con GitHub, el portal en la web para alojar repositorios de código.

Android Studio

Android Studio es el entorno de desarrollo integrado (IDE) oficial para el desarrollo de aplicaciones para Android. Para poder hacer uso de él, será necesario acudir a la página web para descargar el instalador. Una vez descargado, durante la instalación aparecerá una ventana inicial en la que se preguntará qué componentes se quieren instalar:

- Android Studio: obligatorio
- Android SDK (conjunto de herramientas de desarrollo software): opcional
- Android Virtual Device (emulador): opcional

Para este proyecto ha sido necesaria la instalación de los tres componentes.

Una vez instalada, creamos un nuevo proyecto que será nuestra aplicación. Por defecto, Android Studio instala la última versión de Android disponible (actualmente Android 11.0 (R) o API 30) que ha sido necesario modificar en nuestro caso, puesto que la aplicación está destinada a usarse en terminales con versiones anteriores. El uso de las versiones de Android se gestionand desde el SDK Manager (*File* > *Settings* > *Appearance* & *Behavior* > *System Settings* > *Android SDK*).

Además de la versión del sistema operativo, tenemos la posibilidad de ejecutar la aplicación creada tanto en un dispositivo real como en uno virtual (emulador). Este último con la herramienta *AVD Manager*, que te permite la simulación en distintos tipos de dispositivos compatibles (TV, móviles, tablets, etc.). Una vez escogido el dispositivo que queremos simular, escogemos la versión de SO entre las disponibles (descargadas en el SDK Manager). El dispositivo resultante será una imagen idéntica al correspondiente real.

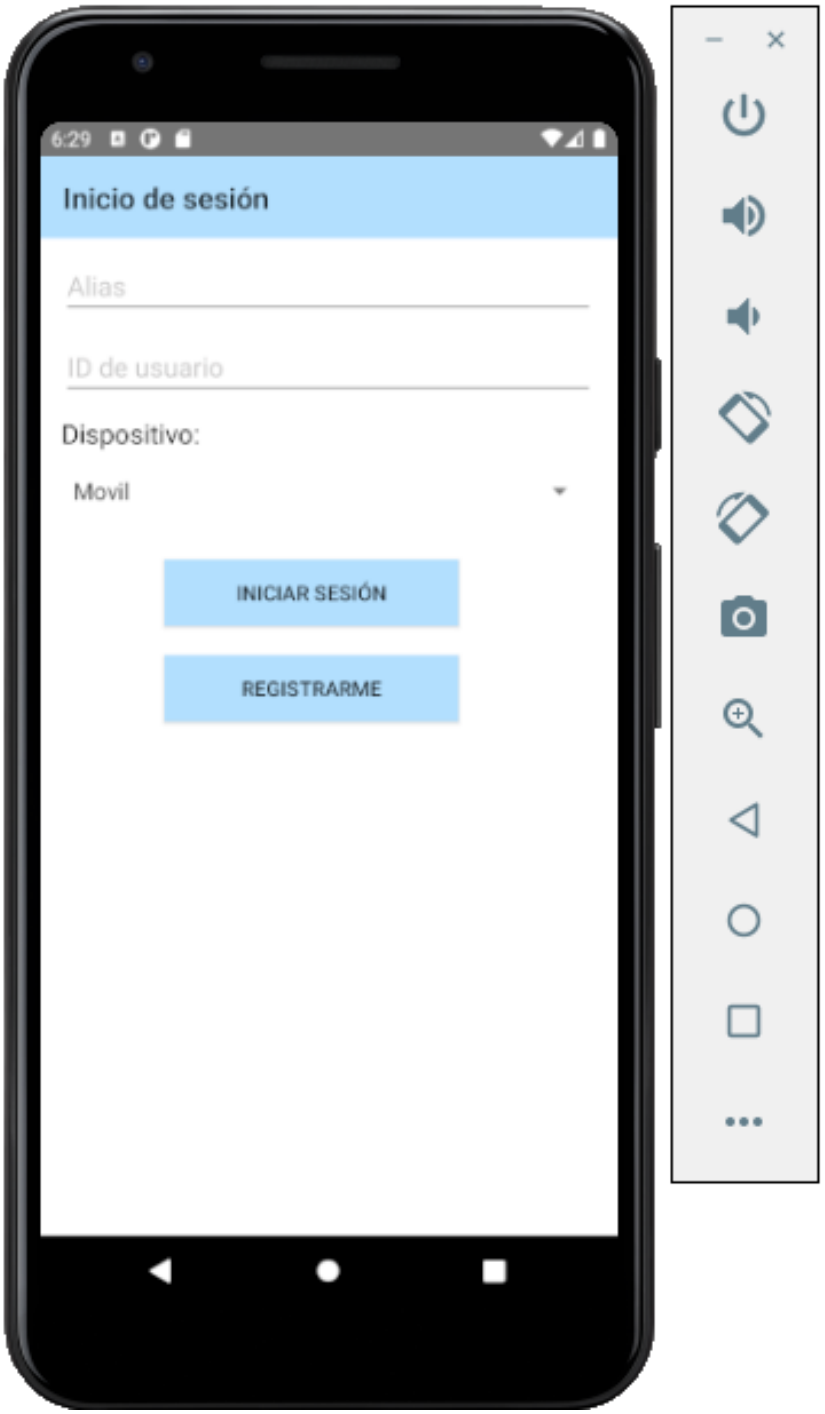


Figura 1: Dispositivo virtual emulado

Como hemos comentado, también podemos probar la aplicación en un dispositivo físico real. Para ello, habrá que conectarlo mediante USB al ordenador donde se ejecute el programa y configurar el móvil en modo desarrollador, seleccionando la opción de *Depuración por USB* en el menú *Opciones de desarrollador*. Este apartado aparecerá en los ajustes únicamente después de activar el modo desarrollador. Cada modelo de dispositivo activa esta opción de manera distinta, por lo que sería necesario buscarlo en la documentación del fabricante y modelo.

Java

Para poder utilizar este lenguaje de programación será necesario descargar el paquete de instalación que incluye lo necesario para ejecutarlo (JRE, *Java Runtime Environment*), así como la máquina virtual donde internamente se ejecuta (JVM, *Java Virtual Machine*). Además, se trata de un lenguaje compilado, por lo que será necesario también el uso de un compilador. Todos estos elementos vienen incluídos en el paquete JDK (*Java Development Kit*), disponibles diferentes versiones en la página oficial de *Oracle* para sistemas operativos Linux, macOS y Windows.

La ejecución del programa se llevará a cabo mediante la ejecución consecutiva de los comandos ***javac*** y ***java***, como se muestra en las siguientes figuras:

```
C:\Users\patry\Documents\TFG>javac ServerConnection.java
```

Figura 2: Compilación con ***javac***

```
C:\Users\patry\Documents\TFG>java ServerConnection
ServerSocket[addr=0.0.0.0/0.0.0.0,localport=49662]
```

Figura 3: Ejecución con ***java***

Wireshark

Este programa es uno de los más empleados para la captura de tráfico de red y está disponible para Windows y macOS. Para poder utilizarlo, habrá que descargar el instalador correspondiente. Uno de los requisitos para capturar tráfico en tiempo real de la red es tener instalado el paquete *Npcap* en el dispositivo de captura, una librería que nos permite hacer sniffing de paquetes en Windows con cualquier analizador de tráfico. Si no está previamente instalado este paquete, el propio instalador de Wireshark nos dará la opción de añadirlo al proceso. Una vez instalado, y para comprobar que funciona correctamente, en la pantalla inicial nos aparecerá una lista con las interfaces de red del dispositivo, y el uso del mismo. En el caso de que inicialmente no aparezca, será necesario abrirlo en modo administrador (*Botón derecho sobre la aplicación > Ejecutar como administrador*).

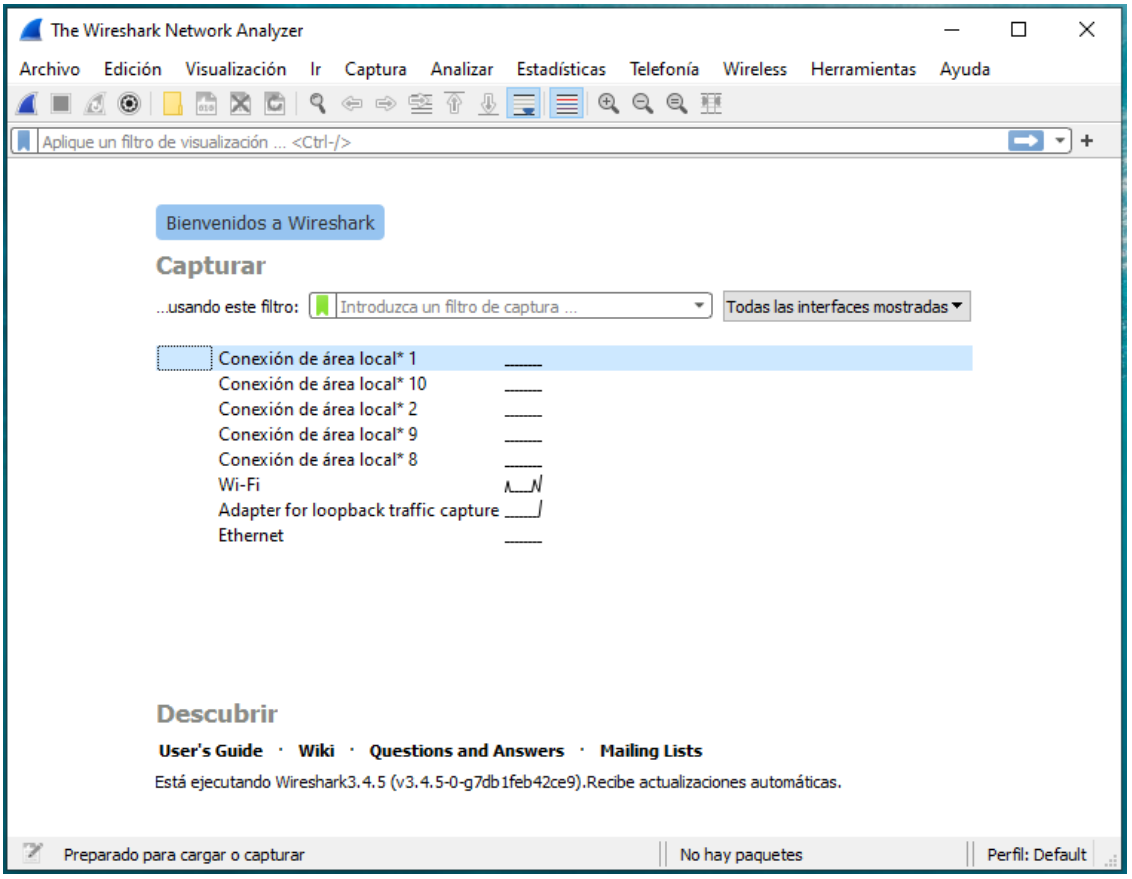


Figura 4: Pantalla de inicio de *Wireshark*

Para empezar a capturar, seleccionamos la interfaz y podremos ver todos los paquetes de esa red, así como filtrar por los protocolos, direcciones IP, etc., que nos interesen.

Anexo 2: Implementación de aplicación móvil Android

En una aplicación Android desarrollada con Java, cada pantalla está formada por dos componentes: una *Activity* o *Fragment*, correspondiente a la parte visual de la pantalla y realizada con lenguaje XML, y una clase en Java que contiene la lógica de la misma (eventos de ratón/teclado realizados sobre ella, recogida de datos de formularios, etc.). Sin embargo, una aplicación no consta únicamente de estos dos elementos, sino que también puede incluir otros como los que se explican a continuación.

A la hora de enviar información, se realiza a partir de un fichero que recopila los datos que se van a mandar al servidor. Para ello, se ha creado una clase que se encarga de escribir esos datos en el fichero correspondiente (*login.txt, register.txt, data.txt, update.txt*) y mandarlo al hilo que que lo enviará al servidor correspondiente. Esta acción se realiza en un hilo aparte por requerimientos del propio SO, que no permite la conexión externa por sockets en el hilo principal. Se ha planteado para cada uno de los tipos de servidor una clase java diferente, en la que se podrá ampliar el número de conexiones si aumentan el número de servidores.

La aplicación dispone de un servicio funcionando en segundo plano, es decir, sin necesidad de que el usuario la tenga abierta, que permite la recopilación de datos y se lanzará al iniciar sesión (o registrarse). Este servicio genera una notificación constante en la barra de estado del dispositivo, siempre que se haya iniciado sesión, de forma que el usuario sea consciente en todo momento del funcionamiento de la app.

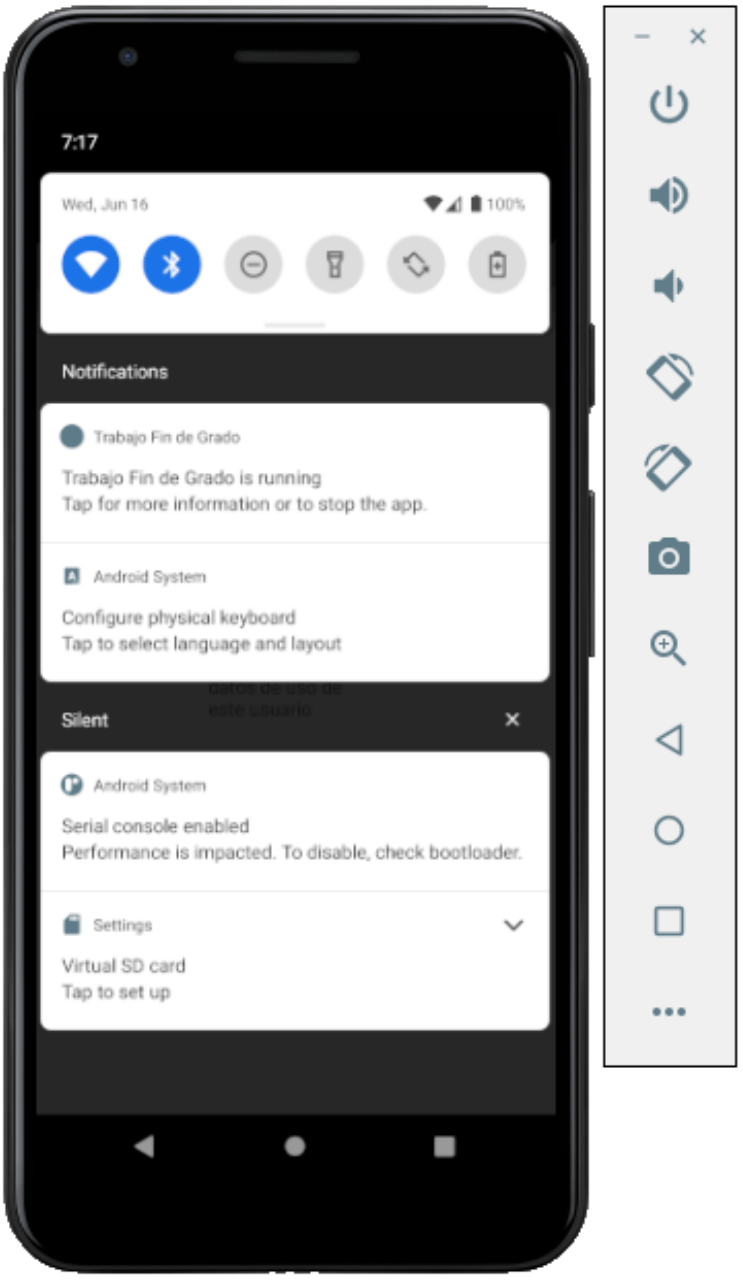


Figura 5: Notificación en la barra de estado

Además de esto, se conecta con un *Broadcast Receiver*, un elemento que recibe y responde a eventos globales, que pueden ser lanzados por una aplicación (por ejemplo, un aviso), o por el propio dispositivo (por ejemplo, desbloqueo de pantalla). En nuestro caso, está preparado para captar los eventos: *ACTION_SCREEN_OFF* (apagado de pantalla), *ACTION_SCREEN_ON* (encendido de pantalla) y *ACTION_USER_PRESENT* (desbloqueo de pantalla).